

Artifactory

User Guide

1. Welcome to Artifactory	4
1.1 Installing Artifactory	6
1.1.1 System Requirements	8
1.1.2 Installing on Linux Solaris or Mac OS	10
1.1.3 Installing on Windows	17
1.1.4 Installing with Docker	20
1.1.4.1 Building Artifactory OSS	24
1.1.4.2 Changing the Database	25
1.2 Upgrading Artifactory	27
1.2.1 Upgrading an Enterprise HA Cluster	37
1.3 Using Artifactory	50
1.3.1 Getting Started	53
1.3.2 General Information	58
1.3.3 Browsing Artifactory	61
1.3.4 Using WebDAV	67
1.3.5 Searching for Artifacts	68
1.3.6 Deploying Artifacts	76
1.3.7 Manipulating Artifacts	83
1.3.8 Updating Your Profile	88
1.3.9 Authentication	93
1.3.10 Artifactory REST API	95
1.3.10.1 Repository Configuration JSON	193
1.3.10.2 Security Configuration JSON	197
1.3.10.3 System Settings JSON	198
1.4 Configuring Artifactory	199
1.4.1 Configuring the Database	204
1.4.1.1 MySQL	206
1.4.1.2 Oracle	210
1.4.1.3 Microsoft SQL Server	211
1.4.1.4 PostgreSQL	215
1.4.2 Configuring the Filestore	216
1.4.3 Checksum-Based Storage	247
1.4.4 Configuring Repositories	250
1.4.4.1 Common Settings	253
1.4.4.2 Local Repositories	255
1.4.4.3 Remote Repositories	259
1.4.4.3.1 Managing Proxies	262
1.4.4.3.2 Advanced Settings	263
1.4.4.4 Smart Remote Repositories	268
1.4.4.5 Virtual Repositories	271
1.4.5 Configuring Security	275
1.4.5.1 Managing Users	280
1.4.5.2 Managing Permissions	286
1.4.5.3 Centrally Secure Passwords	290
1.4.5.4 Master Key Encryption	292
1.4.5.5 Managing Security with LDAP	294
1.4.5.6 Managing Security with Active Directory	297
1.4.5.7 Managing Certificates	304
1.4.5.8 Using a Self-Signed Certificate	306
1.4.5.9 Access Tokens	307
1.4.5.10 Access Log	314
1.4.6 Configuring a Reverse Proxy	315
1.4.6.1 Configuring Apache	320
1.4.6.2 Configuring NGINX	323
1.4.7 Mail Server Configuration	325
1.4.8 Configuration Files	327
1.4.9 Exposing Maven Indexes	331
1.4.10 Clustering Artifactory	333

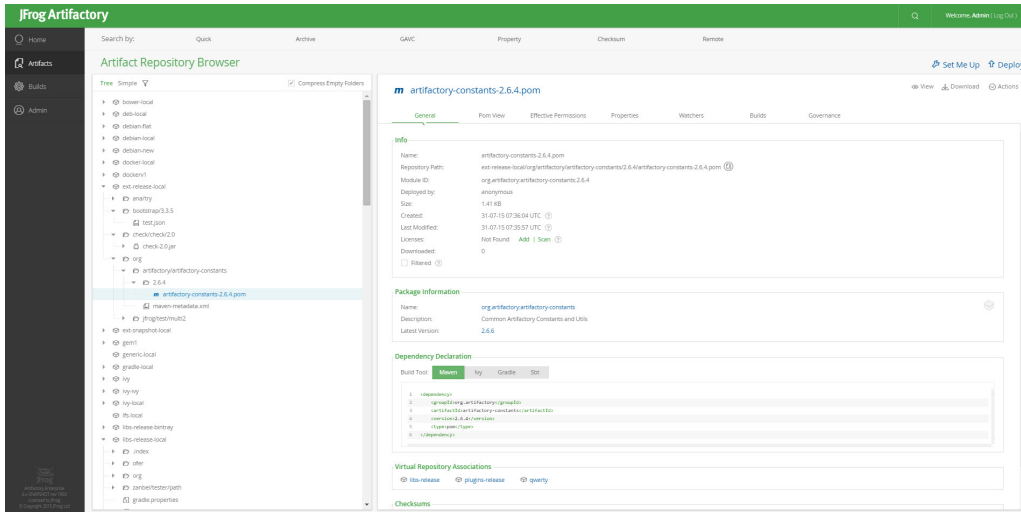
1.5 System Monitoring and Maintenance	335
1.5.1 System Information	335
1.5.2 Monitoring Storage	336
1.5.3 Artifactory Log Files	339
1.5.4 Artifactory JMX MBeans	343
1.5.5 Regular Maintenance Operations	347
1.5.6 Managing Backups	350
1.5.7 Importing and Exporting	353
1.5.8 Managing Disk Space Usage	359
1.5.9 Getting Support	362
1.6 Artifactory High Availability	364
1.6.1 HA Installation and Setup	369
1.6.2 Managing the HA Cluster	382
1.6.3 Migrating Data from NFS	385
1.6.4 Troubleshooting HA	391
1.7 Xray Integration	394
1.8 Bintray Integration	397
1.8.1 Bintray info panel	398
1.8.2 Distribution Repository	398
1.8.3 Deploying Snapshots to oss.jfrog.org	411
1.9 Log Analytics	416
1.10 Artifactory Pro	420
1.10.1 Artifactory Comparison Matrix	422
1.10.2 Pro Features	424
1.10.2.1 Artifactory Query Language	426
1.10.2.2 Atlassian Crowd and JIRA Integration	444
1.10.2.3 Azure Blob Storage	447
1.10.2.4 Black Duck Code Center Integration	448
1.10.2.5 Filestore Sharding	448
1.10.2.6 Filtered Resources	453
1.10.2.7 GPG Signing	455
1.10.2.8 Google Cloud Storage	457
1.10.2.9 LDAP Groups	461
1.10.2.10 License Control	464
1.10.2.11 OAuth Integration	471
1.10.2.12 Properties	480
1.10.2.12.1 Using Properties in Deployment and Resolution	482
1.10.2.13 Repository Layouts	484
1.10.2.14 Repository Replication	492
1.10.2.15 S3 Object Storage	499
1.10.2.16 SAML SSO Integration	501
1.10.2.17 Single Sign-on	507
1.10.2.18 Smart Searches	510
1.10.2.19 SSH Integration	513
1.10.2.20 User Plugins	515
1.10.2.21 Watches	538
1.10.2.22 WebStart and Jar Signing	538
1.10.3 Package Management	540
1.10.3.1 Bower Repositories	542
1.10.3.2 Chef Cookbook Repositories	552
1.10.3.3 CocoaPods Repositories	559
1.10.3.4 Conan Repositories	564
1.10.3.5 Debian Repositories	567
1.10.3.6 Docker Registry	575
1.10.3.6.1 Getting Started with Artifactory as a Docker Registry	587
1.10.3.6.2 Advanced Topics	598
1.10.3.6.3 Working with Docker Content Trust	599
1.10.3.6.4 Using Docker V1	603
1.10.3.7 Git LFS Repositories	612
1.10.3.8 Npm Registry	618
1.10.3.9 NuGet Repositories	628
1.10.3.9.1 Microsoft Symbol Server	638
1.10.3.10 Opkg Repositories	642
1.10.3.11 P2 Repositories	646
1.10.3.12 PHP Composer Repositories	652
1.10.3.13 Puppet Repositories	656
1.10.3.14 PyPI Repositories	671
1.10.3.15 RubyGems Repositories	678
1.10.3.16 SBT Repositories	687
1.10.3.17 Vagrant Repositories	691
1.10.3.18 VCS Repositories	697
1.10.3.19 RPM Repositories	706

1.10.4 Ecosystem Integration	716
1.10.4.1 Maven Repository	717
1.10.4.1.1 Maven Artifactory Plugin	723
1.10.4.2 Working with Gradle	728
1.10.4.2.1 Gradle Artifactory Plugin	733
1.10.4.3 Working with Ivy	741
1.10.5 Build Integration	744
1.10.5.1 Jenkins Artifactory Plug-in	754
1.10.5.1.1 Working With Pipeline Jobs in Jenkins	755
1.10.5.2 TeamCity Artifactory Plug-in	771
1.10.5.2.1 TeamCity Artifactory Plugin - Release Management	784
1.10.5.3 Bamboo Artifactory Plug-in	789
1.10.5.3.1 Bamboo Artifactory Plugin - Release Management	810
1.10.5.4 MSBuild Artifactory Plugin	818
1.10.5.5 VS Team Services Artifactory Plugin	827
1.10.5.6 Using File Specs	836
1.11 Troubleshooting	841
1.12 Known Issues	842
1.13 End of Life	843
1.14 Release Notes	847
1.15 Pivotal Cloud Foundry JFrog Artifactory Tile Release Notes	881

Welcome to Artifactory

Welcome to the JFrog Artifactory User Guide!

JFrog Artifactory is the only Universal Repository Manager supporting all major packaging formats, build tools and CI servers.



This user guide is for **Artifactory 5.0.0** and above.
Click this link to download the latest PDF version of the Artifactory User Guide:

Note that the online version may be more up-to-date.
If you are using Artifactory 4.x.y, please refer to the [Artifactory 4 User Guide](#).

Which Artifactory Do You Need?

Artifactory comes in the following flavors:

Artifactory OSS	<p>Offers powerful features with fine-grained permission control behind a sleek and easy-to-use UI.</p> <p>Using the links in the next column, you can download the Artifactory OSS installation files, or the source files so you can build Artifactory OSS yourself.</p> <p>For more information on building Artifactory OSS, please refer to the Readme file.</p>	Download Download Sources
Artifactory Pro	<p>Exposes a set of professional add-ons, on top of those already available to you from Artifactory Open Source, opening up a whole world of features that empower you to manage your binaries and integrate with industry standard tools in your development and deployment ecosystem.</p>	Download
Artifactory Cloud	<p>JFrog's SaaS-based solution for managing your artifacts and binary repositories in the cloud with the full power of Artifactory Pro behind you and 24/7 SLA-based support.</p>	Register
Artifactory Enterprise	<p>Exposes an enterprise feature set such as cloud storage providers, advanced capabilities for replication, high availability and more</p>	Enterprise Free Trial

To see which version of Artifactory best suits your needs, please see the [Artifactory Features Matrix](#).

How is this Guide Organized?

Installing and Upgrading Artifactory	Learn how to install and upgrade Artifactory on all supported platforms, including detailed system requirements and pre-requisites.
Using Artifactory	Learn how to use Artifactory on a day-to-day basis including creating repositories, deploying, copying and moving artifacts and more.
Configuring Artifactory	Learn how to configure repositories, users, permissions and more.
Artifactory REST API	A detailed specification of Artifactory's extensive REST API letting you automate any process in your development ecosystem.
System Monitoring and Maintenance	Learn how to keep your system free of clutter and operating smoothly.
Artifactory High Availability	Learn how to configure and use Artifactory in a High Availability configuration providing the most stable and secure binary repository available to enterprise users today.
Bintray Integration	Learn how to integrate with JFrog Bintray to completely automate your software development pipeline all the way to distribution.
Artifactory Pro	Learn about all the add-ons that let Artifactory work seamlessly with packaging formats such as Docker, NuGet, Vagrant, RubyGems and more, as well as with all major CI servers and build tools
Release Notes	Learn about the changes that came with each release of Artifactory.

Distributing Software Through Bintray

Bintray is JFrog's universal distribution platform.

Through tight integration, you can use Artifactory to push artifacts directly to your repositories in Bintray, search through your Bintray repositories and more to fully automate your software distribution process. For more details, please refer to [Bintray Integration](#).

For more details on how to use Bintray, please refer to the [Bintray User Guide](#).











Page Contents

- [Welcome to the JFrog Artifactory User Guide!](#)
- [Which Artifactory Do You Need?](#)
- [How is this Guide Organized?](#)
- [Distributing Software Through Bintray](#)

Quick Links

[Artifactory REST API](#)[Docker Registry](#)[Release Notes](#)

Recently Updated

-  [Artifactory REST API](#)
29 minutes ago • updated by [Rami Honig](#) • [view change](#)
-  [Artifactory REST API](#)
about 9 hours ago • updated by [Shlomi Kriheli](#) • [view change](#)
-  [HA Installation and Setup](#)
yesterday at 7:39 PM • updated by [Rami Honig](#) • [view change](#)
-  [User Plugins](#)
yesterday at 2:44 PM • updated by [Rami Honig](#) • [view change](#)
-  [JFrog Platform 5.7 - DONE](#)
yesterday at 12:10 PM • updated by [Rami Honig](#) • [view change](#)
-  [Working With Pipeline Jobs in Jenkins](#)
yesterday at 10:53 AM • updated by [Eyal Ben Moshe](#) • [view change](#)
-  [VCS Repositories](#)
yesterday at 10:00 AM • updated by [Rami Honig](#) • [view change](#)
-  [Release Notes](#)
Nov 27, 2017 • updated by [Rami Honig](#) • [view change](#)
-  [Release Notes 5.6.2](#)
Nov 27, 2017 • updated by [Rami Honig](#) • [view change](#)
-  [Getting Started with Artifactory as a Docker Registry](#)
Nov 27, 2017 • updated by [Rami Honig](#) • [view change](#)

Installing Artifactory

Overview

This section provides a guide on the different ways you can install and configure Artifactory.

Installing Artifactory HA

There are different instructions for installing Artifactory HA
If you are installing an Artifactory HA cluster, please refer to [HA Installation and Setup](#).

If you follow the instructions on this page **for an installation of Artifactory HA**, your HA cluster will not work.

System Requirements

Before you install Artifactory please refer to [System Requirements](#) for information on supported platforms, supported browsers and other requirements.

Installation

The installation procedure involves the following main steps:

1. [Installing Artifactory](#)
2. [Configuring the database](#)
3. [Configuring the filestore](#)
4. [Configuring an HTTP Server \(Optional\)](#)

Installing Artifactory

For detailed instructions, visit one of the following platform-specific pages:

- [Installing on Linux, Solaris or Mac OS](#)
- [Installing on Windows](#)
- [Installing with Docker](#)

Configuring the Database

Artifactory comes with an embedded Derby Database out-of-the-box which it is pre-configured to use, however, for better performance and to reuse existing infrastructures you may have, you can configure Artifactory to work with alternative supported databases.

For details please refer to [Configuring the Database](#).

Configuring the Filestore

By default, Artifactory is configured to use the local file system as its filestore. Artifactory supports a variety of additional filestore configurations to meet a variety of needs for binary storage providers, storage size and redundancy. For details, please refer to [Configuring the Filestore](#).

Configuring an HTTP Server

You can run Artifactory with one of the supported HTTP servers set up as a front end. For details please refer to [Configuring a Reverse Proxy](#).

Page Contents

- [Overview](#)
- [Installing Artifactory HA](#)
- [System Requirements](#)
- [Installation](#)
 - [Installing Artifactory](#)
 - [Configuring the Database](#)
 - [Configuring the Filestore](#)
 - [Configuring an HTTP Server](#)
- [Directory Structure](#)
- [Default Admin User](#)
- [Watch the Screencast](#)
- [Troubleshooting](#)
 - [Artifactory Does Not Start Up](#)

Read more

- [System Requirements](#)
- [Installing on Linux Solaris or Mac OS](#)
- [Installing on Windows](#)
- [Installing with Docker](#)

Directory Structure

After installing Artifactory, the `$ARTIFACTORY_HOME` directory will contain the following directory structure (the `$ARTIFACTORY_HOME` directory location depends on your installation type):

<i>File/Folder</i>	<i>Description</i>
<code>access</code>	The home directory of the bundled JFrog Access. More details in the Access Tokens page.
<code>access/etc/keys</code>	JFrog Access keys. More details in the Access Tokens page.

<i>logs</i>	Artifactory log files (general, access, request etc.)
<i>etc</i>	Configuration files
<i>etc/plugins</i>	Custom Groovy user plugins.
<i>etc/security</i>	Global security related files (configuring global encryption key, PGP signing key etc.).
<i>etc/ui</i>	Manually uploaded custom UI logos.
<i>data/derby</i>	The Derby database (only present when using Derby).
<i>data/filestore</i>	The checksum based storage of binaries when using the default filesystem storage.
<i>data/tmp/work</i>	Directory to save temporary files which Artifactory generates.
<i>data/tmp/artifactory-uploads</i>	Directory to save files uploaded using the Web UI.
<i>bin</i>	Artifactory startup/shutdown scripts.
<i>tomcat</i>	The default tomcat directory bundled with Artifactory.
<i>tomcat/work</i>	The tmp directory tomcat and the JVM uses (Tomcat automatically assigns it to a java system environment variable as <code>java.io.tmpdir</code>)
<i>tomcat/logs</i>	Additional Tomcat log files
<i>misc</i>	Configuration files used as examples for different databases and servlet containers.
<i>backup</i>	The default backup directory Artifactory uses for system wide and repository backup.
<i>webapps</i>	Contains the Artifactory WAR file and the Access WAR file used by the bundled Tomcat distribution. We strongly recommend keeping both these files in the same bundled Tomcat.

Default Admin User

Once installation is complete, Artifactory has a default user with admin privileges predefined in the system:

User: admin

Password: password

Change the admin password

We strongly recommend changing the admin password as soon as installation is complete.

Watch the Screencast

Troubleshooting

Artifactory Does Not Start Up

✓ [There are no log file entries in \\$ARTIFACTORY_HOME/logs/artifactory.log](#)

Cause	An exception was thrown (possibly by your servlet container) before Artifactory loaded its logging mechanism.
Resolution	Check your servlet container's localhost.log file. For more information, please refer to Artifactory Log Files .

System Requirements

Supported Platforms

Artifactory has been tested and verified on Linux, Windows (Vista and higher), Solaris and Mac OS X. You should be able to run Artifactory on other platforms, but these have not been tested.

JDK

You must run Artifactory with **JDK 8**, preferably JDK 8 update 45 and above.

You can download the latest JDK from the [Oracle Java SE Download Site](#).

JAVA_HOME and JRE_HOME

Make sure your JAVA_HOME environment variable correctly points to your JDK 8 installation.

If you also have JRE_HOME defined in your system, this will take precedence over JAVA_HOME and therefore you need to either point JRE_HOME to your JDK 8 installation, or remove the JRE_HOME definition.

Page Contents

- [Supported Platforms](#)
- [JDK](#)
- [JVM Memory Allocation](#)
- [Browsers](#)
- [Recommended Hardware](#)
 - [Working with Very Large Storage](#)
- [High Availability Configuration](#)
- [Database Requirements](#)
- [Servlet Containers](#)

JVM Memory Allocation

While not a strict requirement, we recommend that you modify the JVM memory parameters used to run Artifactory.

You should reserve at least 512MB for Artifactory, and the recommended values for JVM parameters are as follows:

Recommended JVM parameters

The larger your repository or number of concurrent users, the larger you need to make the -Xms and -Xmx values accordingly.

Recommended values are:

```
-server -Xms512m -Xmx2g -Xss256k -XX:+UseG1GC
```

To set your JVM parameters according to your platform, please refer to the corresponding instructions for [Linux](#), [Solaris or Mac](#), or [Windows](#).

Browsers

Artifactory has been tested with the latest versions of Google Chrome, Firefox, Internet Explorer and Safari.

Recommended Hardware

The following table provides hardware recommendations for a single server machine:

Number of developers	OS/JVM	Processor	*Memory (RAM) for JVM Heap	Storage
1 - 20	64 bit	4 cores	4GB	Fast disk with free space that is at least 3 times the total size of stored artifacts
20 - 100	64 bit	4 cores	8GB	Fast disk with free space that is at least 3 times the total size of stored artifacts
100 - 200	64 bit	8 cores (16 cores recommended)	12GB	Fast disk with free space that is at least 3 times the total size of stored artifacts (backup SAN recommended)
200+	64 bit	Please contact JFrog support for a recommended setup.		

*Memory (RAM) for JVM Heap

This specifies the amount of memory that Artifactory requires from the JVM heap. The server machine should have enough additional memory to run the operating system and any other processes running on the machine.

Build machine

For the purposes of this table, a build machine is considered equivalent to 10 developers

Working with Very Large Storage

In most cases, our recommendation is for storage that is at least 3 times the total size of stored artifacts in order to accommodate [system backups](#). However, when working with a very large volume of artifacts, the recommendation may vary greatly according to the specific setup of your system.

Therefore, when working with over **10 Tb** of stored artifacts, please contact [JFrog support](#) who will work with you to provide a recommendation for storage that is customized to your specific setup.

High Availability Configuration

If you are running Artifactory in a High Availability configuration, to maintain high system performance in case of single or multiple server crash, we recommend following the [recommended hardware](#) guidelines above for each of the HA server instances. For more details, please refer to [Artifactory High Availability](#).

Database Requirements

To avoid network latency issues when reading and writing artifacts data, we strongly recommend creating the database either on a machine that is network close (latency well below 1 ms) to the machine on which Artifactory is running (database engine and storage) with fast storage. This recommendation is critical when using [fullDb](#) (whereby files are served from database BLOBs) and the file system cache is small.

For supported databases and more details, please refer to [Configuring the Database](#).

Servlet Containers

Artifactory should be run with its bundled Tomcat 8 servlet container.

From version 5.0, Artifactory is bundled with Tomcat version 8.0.39.

Installing on Linux Solaris or Mac OS

Overview

Make sure you have reviewed the overall installation process

Before you proceed with the instructions on this page, make sure you have reviewed the whole installation procedure as described in [Installing Artifactory](#).

This page describes how to install Artifactory on Linux, Solaris or Mac OS.

The procedure for all these platforms is identical, so for the sake of clarity the rest of this page will refer to Linux only.

You can install Artifactory on your Linux system in one of the following ways:

- [Manual Installation](#)
- [Service Installation](#)
- [RPM or Debian Installation](#)
- [As a Docker Image](#)

Running as root to install Artifactory as a service or RPM distribution

To install Artifactory as a service or RPM distribution you must have root privileges.

To run as root either execute the following command:

```
su -
```

or precede all commands with `sudo` (e.g. `sudo service artifactory start`)

If you are unable to get root privileges please contact your system administrator.

Configuring Your Database and Filestore

Once you have completed installing Artifactory, make sure you configure its database and filestore according to your preference. For details, please refer to [Configuring the Database](#) and [Configuring the Filestore](#).

Page Contents

- [Overview](#)
 - [Configuring Your Database and Filestore](#)
- [Requirements](#)
 - [Setting JAVA_HOME](#)
 - [Setting Java Memory Parameters](#)
- [Manual Installation](#)
 - [Installing Artifactory](#)
 - [Running Artifactory](#)
- [Service Installation](#)
 - [Installing Artifactory](#)
 - [Running Artifactory](#)
 - [Using systemd](#)
 - [Using init.d](#)
 - [Checking the Artifactory Log](#)
- [RPM or Debian Installation](#)
 - [Managed Files and Folders](#)
 - [Installing Artifactory](#)
 - [Running Artifactory](#)
 - [Backup and Recover](#)
- [Running with Docker](#)

- The ARTIFACTORY_HOME Folder
- Accessing Artifactory

Requirements

Setting JAVA_HOME

As mentioned in the section on [System Requirements](#), make sure that your `JAVA_HOME` environment variable is correctly set to your JDK installation.

Setting Java Memory Parameters

While not a strict requirement, it is recommended to modify the JVM memory parameters used to run Artifactory.

If you can reserve at least 512MB for Artifactory, the recommended values for JVM parameters are:

Recommended minimal JVM parameters

The larger your repository or number of concurrent users, the larger you need to make the `-Xms` and `-Xmx` values accordingly. Recommended minimal values are:

```
-server -Xms512m -Xmx2g -Xss256k -XX:+UseG1GC
```

For more recommendations about your hardware configuration (especially the `-Xmx` parameter), please refer to [Recommended Hardware](#).

Where you set your JVM parameters depends on how you are running Artifactory:

- For a [manual installation](#), modify `JAVA_OPTIONS` in `$ARTIFACTORY_HOME/bin/artifactory.default`.
- For a [service installation](#), modify `JAVA_OPTIONS` in `$ARTIFACTORY_HOME/etc/default` (you will need to stop and then restart the service after making the modification)
- For an [RPM or Debian installation](#), modify `JAVA_OPTIONS` in `/etc/opt/jfrog/artifactory/default`

Manual Installation

Installing Artifactory

To install Artifactory manually, simply unzip the Artifactory download file to a location on your file system. This will be your `$ARTIFACTORY_HOME` location.

No further action is needed.

Don't forget to modify your [JVM parameters](#) as needed by setting `JAVA_OPTIONS` in `$ARTIFACTORY_HOME/bin/artifactory.default`.

Running Artifactory

You can run Artifactory manually to see its behavior by directly executing:

```
$ARTIFACTORY_HOME/bin/artifactory.sh
```

The console is locked on the Artifactory process and you can stop it cleanly with `Ctrl+C`.

To directly run Artifactory as a daemon process, using the environment variables of the shell you are currently in, execute the following script:

```
$ARTIFACTORY_HOME/bin/artifactoryctl start
```


Startup time

Depending on your system performance it may take Artifactory several seconds to start up. If you try to access Artifactory through your browser while it is starting up, within a few seconds it will provide a notification that it is in the startup process.

Using the same script, you can check if Artifactory is running and display its process id, or stop it using:

Checking if Artifactory is running or stopping it

```
$ARTIFACTORY_HOME/bin/artifactoryctl check | stop
```

To run the Artifactory UI see [Accessing Artifactory](#).

Service Installation

Artifactory is packaged as a zip file with a bundled Tomcat, and a complete install script that can be used to install it as a service running under a custom user.

Permissions

When running Artifactory as a service, the installation script creates a user called `Artifactory` which must have run and execute permissions on the installation directory.

Therefore it is recommended to extract the Artifactory download file into a directory that gives run and execute permissions to all users such as `/opt`

Not supported on Mac OS

The service Installation is currently supported only on Linux and Solaris. It is not supported with Mac OS.

Don't forget to modify your [JVM parameters](#) as needed by setting `JAVA_OPTIONS` in `$ARTIFACTORY_HOME/etc/default`.

You need to reinstall the service for your changes to take effect.

Installing Artifactory

To install Artifactory as a service, browse to your `$ARTIFACTORY_HOME/bin` directory and execute the following command as root:

Running the installation script as root

```
$ARTIFACTORY_HOME/bin/installService.sh [USER [GROUP]]
```

The following table describes the sequence of commands performed by the install script:

User creation	<p>Creates a default user named <code>artifactory</code> (<code>\$ARTIFACTORY_USER</code>). You can change the default user by editing the <code>\$ARTIFACTORY_USER</code> value in <code>/etc/opt/jfrog/artifactory/default</code>.</p> <p>You can also optionally run the install script to start the Artifactory service under a different user by passing in the user name as the first parameter. If you include the user name, you may also optionally include a group.</p>
etc config	<p>Creates the folder <code>/etc/opt/jfrog/artifactory</code>, copies the configuration files there and creates a soft link in <code>\$ARTIFACTORY_HOME/etc</code></p>

etc default	<p>Creates the file <code>/etc/opt/jfrog/artifactory/default</code> containing the main environment variables needed for Artifactory to run: <code>JAVA_HOME</code>, <code>ARTIFACTORY_USER</code>, <code>ARTIFACTORY_HOME</code>, <code>JAVA_OPTIONS</code>,...</p> <p>The <code>/etc/opt/jfrog/artifactory/default</code> is included at the top of <code>artifactoryctl</code> and can include any settings.</p> <p>To modify your JVM parameters modify <code>JAVA_OPTIONS</code> in <code>/etc/opt/jfrog/artifactory/default</code></p>
systemd or init.	<p>If you are running on a Linux distribution that supports <code>systemd</code>, the install script will use it to install Artifactory - otherwise <code>init.d</code> will be used.</p> <p>If <code>systemd</code> is supported, the install script copies the service script file <code>artifactory</code> to <code>/etc/systemd/system/artifactory.service</code></p> <p>If <code>systemd</code> is not supported and <code>init.d</code> is used, the install script copies the service script file <code>artifactory</code> to <code>/etc/init.d/artifactory</code></p>
Logs folder	<p>Creates the folder <code>\$ARTIFACTORY_HOME/logs</code>, makes it writable for the user <code>ARTIFACTORY_USER</code> and creates a soft link <code>\$ARTIFACTORY_HOME/logs/catalina</code>.</p> <p>The <code>\$ARTIFACTORY_HOME/tomcat/logs</code> folder is linked to <code>\$ARTIFACTORY_HOME/logs/catalina</code>.</p>
Backup folder	<p>Creates the folder <code>\$ARTIFACTORY_HOME/backup</code>, so you must create a link if you want this folder to point to a different place (such as <code>/var/backup/artifactory</code> for example). The script makes <code>\$ARTIFACTORY_HOME/backup</code> writable for the user <code>ARTIFACTORY_USER</code>.</p>
Data folder	<p>Creates the folder <code>\$ARTIFACTORY_HOME/data</code>, so you must create a link if you want this folder to point to somewhere else. The script makes it writable for the user <code>ARTIFACTORY_USER</code>.</p>
chkconfig calls	<p>The script calls <code>add</code> and <code>list</code> (you can see the output), and then activates the Artifactory service</p>

Running Artifactory

To start or stop Artifactory as a service you must be running as root. The command you use depends on whether the Artifactory service was installed using `systemd` or `init.d`.

Using systemd

Start or stop Artifactory using:

```

Artifactory installed with systemd
systemctl <start | stop> artifactory.service

```

Checking the status of the Artifactory service

Once Artifactory is correctly installed, you can check if it is running with:

```
systemctl status artifactory.service
```

If Artifactory is running, you should see its pid.

If Artifactory is not running you will see a list of environment variables used by the service.

Using init.d

Start or stop Artifactory using

```

Artifactory installed with init.d
service artifactory <start | stop>

```

Checking the status of the Artifactory service

Once Artifactory is correctly installed, you can check if it is running with:

```
service artifactory check
```

If Artifactory is running, you should see its pid.

If Artifactory is not running you will see a list of environment variables used by the service.

Checking the Artifactory Log

You can check the Artifactory log to see the status of the service using:

```
tail -f $ARTIFACTORY_HOME/logs/artifactory.log
```

RPM or Debian Installation

Artifactory can also be installed from an RPM or Debian distribution on Red Hat compatible Linux distributions.

The installation package creates a dedicated user, installs a stripped-down distribution of the Apache Tomcat container configured for Artifactory (on port 8081), and registers this Tomcat as a service (but does not start it immediately).

This package effectively replaces the different setup scripts included with the Artifactory Zip distribution.

Managed Files and Folders

When installed from an RPM distribution, Artifactory retains the FHS (Filesystem Hierarchy Standard) format:

File/Folder	Location	Ownership
Artifactory home	<code>/var/opt/jfrog/artifactory</code>	artifactory
Artifactory etc	<code>/etc/opt/jfrog/artifactory</code>	artifactory
Artifactory logs	<code>/var/opt/jfrog/artifactory/logs</code>	artifactory
Artifactory env variables	<code>/etc/opt/jfrog/artifactory/default</code>	artifactory
Tomcat home	<code>/opt/jfrog/artifactory/tomcat</code>	artifactory (root for sub dirs)
Artifactory startup script	<code>/etc/init.d/artifactory</code>	root
Artifactory binary	<code>/opt/jfrog/artifactory</code>	root

Installing Artifactory

To install Artifactory from an RPM or Debian distribution **you must be running as root** and can use the corresponding commands below:

✓ Installing Artifactory Pro from an RPM distribution...

```
wget https://bintray.com/jfrog/artifactory-pro-rpms/rpm -O bintray-jfrog-artifactory-pro-rpms.repo
sudo mv bintray-jfrog-artifactory-pro-rpms.repo /etc/yum.repos.d/
sudo yum install jfrog-artifactory-pro
```

✓ Installing Artifactory OSS from an RPM distribution

```
wget https://bintray.com/jfrog/artifactory-rpms/rpm -O bintray-jfrog-artifactory-rpms.repo
sudo mv bintray-jfrog-artifactory-rpms.repo /etc/yum.repos.d/
sudo yum install jfrog-artifactory-oss
```

✓ Installing Artifactory Pro from a Debian distribution...

```
echo "deb https://jfrog.bintray.com/artifactory-pro-debs {distribution} {components}" | sudo tee -a /etc/apt/sources.list
```

Note: If you are unsure, components should be "main." To determine your distribution, run `lsb_release -c`

Example: `echo "deb https://jfrog.bintray.com/artifactory-pro-debs xenial main" | sudo tee -a /etc/apt/sources.list`

`curl https://bintray.com/user/downloadSubjectPublicKey?username=jfrog | sudo apt-key add -`

`sudo apt-get update`

`sudo apt-get install jfrog-artifactory-pro`

▼ Installing Artifactory OSS from a Debian distribution...

`echo "deb https://jfrog.bintray.com/artifactory-debs {distribution} {components}" | sudo tee -a /etc/apt/sources.list`

Note: If you are unsure, components should be "main." To determine your distribution, run `lsb_release -c`

Example: `echo "deb https://jfrog.bintray.com/artifactory-debs xenial main" | sudo tee -a /etc/apt/sources.list`

`curl https://bintray.com/user/downloadSubjectPublicKey?username=jfrog | sudo apt-key add -`

`sudo apt-get update`

`sudo apt-get install jfrog-artifactory-oss`

JVM parameters

Make sure to modify your [JVM parameters](#) by modifying `JAVA_OPTIONS` in `/etc/opt/jfrog/artifactory/default` as appropriate for your installation.

Running Artifactory

To start or stop Artifactory **you must be running as root** and can use the following command:

```
service artifactory start | stop
```

Checking the status of the Artifactory service

Once Artifactory is correctly installed, you can check if it is running with:

```
service artifactory check
```

If Artifactory is running, you should see its pid.

If Artifactory is not running you will see a list of environment variables used by the service.

You can also check the Artifactory log with:

```
tail -f $ARTIFACTORY_HOME/logs/artifactory.log
```

When installing from an RPM distribution, Artifactory is generally started as `root` and will `su` internally to the `$ARTIFACTORY_USER` user.

Security

For reasons of security, it is not recommended to leave the `$ARTIFACTORY_USER` variable undefined with Artifactory running as the current user, especially if the current user is `root`.

Backup and Recover

When uninstalling an RPM distribution of Artifactory, it will save the `$ARTIFACTORY_HOME` folder and create a backup folder at `/var/opt/jfrog/` while preserving symbolic links to remote filestores.

After installing a new instance of Artifactory, you can recover the configuration and filestore from this backup by running the script `$ARTIFACTORY_BINARY/bin/recover.backup.sh`.

Working with an external database

This process does not back up an external database, but rather its definitions in Artifactory. Therefore, when working with an external database, a manual dump should be performed before uninstalling the RPM, and then imported when starting the new installation.

Installing/Upgrading on a new machine

The Backup and Recover described above will only work if you are re-installing the RPM on the same machine. If you are installing or upgrading the RPM on a new machine you will need to use Import as described in the section on [Upgrading Artifactory](#).

Running with Docker

From Version 3.6, Artifactory may be pulled as a Docker Image. For full details, please refer to [Installing with Docker](#).

The `ARTIFACTORY_HOME` Folder

It is important to know where your Artifactory home folder is located, since this folder stores your configurations and important repository data.

When Artifactory runs for the first time, it sets up a default configuration and creates all necessary files and folders under the `ARTIFACTORY_HOME` folder.

The default location of `ARTIFACTORY_HOME` is `{user.home}/.artifactory`.

To run Artifactory with the home folder set to a different location on the file system (particularly when installing Artifactory on a production server), either:

- Start the Tomcat virtual machine with `-Dartifactory.home=<your preferred Artifactory home folder location>`
- or -
- Set an `ARTIFACTORY_HOME` environment variable pointing to your preferred location before running the installation.

Artifactory creates the home folder on startup if it does not already exist.

Permissions on the Artifactory Home Folder

Make sure that the user running the Tomcat has write permissions on the Artifactory home folder.

Accessing Artifactory

Artifactory can be accessed using the following URL:

`http://SERVER_DOMAIN:8081/artifactory`

For example, if you are testing on your local machine you would use: `http://localhost:8081/artifactory`

Installing on Windows

Overview

Make sure you have reviewed the overall installation process

Before you proceed with the instructions on this page, make sure you have reviewed the whole installation procedure as described in [Installing Artifactory](#).

There are three ways to install Artifactory on your Windows system:

- [Manual Installation](#)
- [Service Installation](#)
- [As a Docker Image](#)

Unzip the Artifactory download file to a location on your file system.

This will be your `%ARTIFACTORY_HOME%` location.

Define this location as an environment variable called `ARTIFACTORY_HOME`.

Configuring Your Database and Filestore

Once you have completed installing Artifactory, make sure you configure its database and filestore according to your preference. For details, please refer to [Configuring the Database](#) and [Configuring the Filestore](#).

Page Contents

- Overview
 - [Configuring Your Database and Filestore](#)
- Requirements
 - [Setting JAVA_HOME](#)
 - [Setting Java Memory Parameters](#)
- [Manual Installation](#)
- [Service Installation](#)
 - [Running Artifactory](#)
- [Running with Docker](#)
- [Accessing Artifactory](#)

Requirements

Setting JAVA_HOME

As mentioned in the section on [System Requirements](#), make sure that your `JAVA_HOME` environment variable is correctly set to your JDK installation.

Troubleshooting

Note that normally the installation path to your `JAVA_HOME` might include a space, e.g. `c:\Program Files (x86)\java\jdk`, this might cause an issue and result in error "Files was unexpected at this time", in which case you will need to replace the `Program Files (x86)` with `PROGRA~2` and make sure there are no spaces in the path.

Setting Java Memory Parameters

While not a strict requirement, it is recommended to modify the JVM memory parameters used to run Artifactory.

This is done by modifying the `JAVA_OPTIONS` variable in `artifactory.bat`, for a [manual installation](#), or the `JOPTS` variable in `installService.bat` when running Artifactory as a [service](#).

For your changes to take effect you need to stop Artifactory and then rerun the modified file.

If you can reserve at least 512MB for Artifactory, the recommended values for JVM parameters are as follows:

Recommended JVM parameter settings

The larger your repository or number of concurrent users, the larger you need to make the `-Xms` and `-Xmx` values accordingly.

Recommended values are:

```
-server -Xms512m -Xmx2g -Xss256k -XX:+UseG1GC
```

Manual Installation

Browse to `%ARTIFACTORY_HOME%\bin` and execute the file `artifactory.bat`. This script searches for the Java executable and runs Artifactory's main class.

Security settings

Depending on the security settings under Windows, you might need to run `artifactory.bat` using 'Run as administrator'

Don't forget to modify your **JVM parameters** as needed by setting `JAVA_OPTIONS` in `ARTIFACTORY_HOME/bin/artifactory.bat`.

To test your installation see [Accessing Artifactory](#).

Service Installation

Artifactory makes use of the [Apache Commons Procrun](#) components allowing you to install the application as a Windows Service.

To run Artifactory as a Windows service, browse to `%ARTIFACTORY_HOME%\bin`, and execute the file `InstallService.bat`.

By editing `InstallService.bat`, you can modify default properties such as `JOPTS` and the log directory.

For your changes to take effect you need to stop the currently running Artifactory service and run `InstallService.bat` again once you have completed your modifications.

To test your installation see [Accessing Artifactory](#).

Security

Windows 8 implements strict User Account Control (UAC). You must either disable UAC or right-click on `cmd.exe` and select "Run as administrator" in order to run this script.

Running on 32 bit Windows

If you are running a 32 bit version of Windows you need to do the following:

- Download the latest version of the [Apache Commons Daemon](#).
- Take `prunsvr.exe` from the downloaded archive and rename it to `artifactory-service.exe`
- Replace the current `artifactory-service.exe` found in your `%ARTIFACTORY_HOME%\bin` directory

Don't forget to modify your **JVM parameters** as needed by setting `JAVA_OPTIONS` in `ARTIFACTORY_HOME/etc/default`.

You need to reinstall the service for your changes to take effect.

Running Artifactory

After installing Artifactory you need to start the service.

To start or stop Artifactory as a service you can use the following command in a **Command Prompt** window:

Starting and stopping the Artifactory service

```
sc start|stop Artifactory
```

Checking the status of the Artifactory service

Once Artifactory is correctly installed, you can check if it is running with:

```
sc query Artifactory
```

You can also use any standard text editor to view the artifactory log data found in `ARTIFACTORY_HOME/logs/artifactory.log`

Running with Docker

From Version 3.6, Artifactory may be pulled as a Docker Image. To run Artifactory in a Docker container on a Windows system, you first need to install [boot2docker](#).

For full details, please refer to [Installing with Docker](#).

Accessing Artifactory

Artifactory can be accessed using the following URL:

http://SERVER_DOMAIN:8081/artifactory.

For example, if you are testing on your local machine you would use: <http://localhost:8081/artifactory>

Installing with Docker

Overview

Make sure you have reviewed the overall installation process

Before you proceed with the instructions on this page, make sure you have reviewed the whole installation procedure as described in [Installing Artifactory](#).

Artifactory Docker images can be pulled from Bintray and run as a Docker container.

To do this, you need to have Docker client properly installed and configured on your machine. For details about installing and using Docker, please refer to the [Docker documentation](#).

Running with Docker for Artifactory 4.x

Artifactory as a Docker container has been completely redesigned in version 5.0. If you are running previous versions of Artifactory, please refer to [Running with Docker](#) in the Artifactory 4.x User Guide

Docker Compose

The way we recommend running Artifactory on Docker is to orchestrate your setup using [Docker Compose](#). This will ensure you have all the required services specified in a single YAML file with pre-configured parameters.

Using Docker Compose

To setup an Artifactory environment made of multiple containers (for example, a database, an Nginx load balancer and Artifactory each running in a different container), you can use `docker-compose`.

For more details on Docker Compose, please refer to the [Docker documentation](#).

Artifactory OSS, **Artifactory Pro** and **Artifactory HA** can all be run using Docker Compose. For detailed documentation and sample Compose files showing a variety of ways to setup Artifactory with Docker Compose, please refer to the [artifactory-docker-examples](#) repository on GitHub.

Page contents

- Overview
- Using Docker Compose
- Artifactory on Docker
 - Pulling the Artifactory Docker Image
 - Running an Artifactory Container
- Managing Data Persistence
 - Using Host Directories
 - Using a Docker Named Volume
- Upgrading Artifactory
- Running Artifactory With a Different Database
- Building Artifactory OSS From Sources
- Accessing Artifactory
- Troubleshooting Docker
 - Container State
 - Logs
 - Connect to a Running Container
 - Run an Alternate Entrypoint
- Watch the Screencast

Read more

- [Building Artifactory OSS](#)
- [Changing the Database](#)

Artifactory on Docker

Running Artifactory as a container is simple and straightforward, and involves the following basic steps:

- [Pulling the Artifactory Docker Image](#)
- [Running the Artifactory Container](#)

Since the Artifactory instance running in a Docker container is mutable, all data and configuration files will be lost once the container is

removed. If you want your data to persist (for example when upgrading to a new version), you should also follow the next step.

- [Managing Data Persistence](#)

Pulling the Artifactory Docker Image

The Artifactory Docker image may be pulled from Bintray by executing the corresponding Docker command below depending on whether you are pulling Artifactory OSS or Artifactory Pro:

Pulling the Artifactory Pro Docker Image

```
docker pull docker.bintray.io/jfrog/artifactory-pro:latest
```

or

Pulling the Artifactory OSS Docker Image

```
docker pull docker.bintray.io/jfrog/artifactory-oss:latest
```

Running an Artifactory Container

You can list the Docker images you have downloaded using the **docker images** command, which should display something like the following output:

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID
docker.bintray.io/jfrog/artifactory-pro	latest	da70b82904e7
2 days ago	861.5 MB	
...		

To start an Artifactory container, use the corresponding command below according to whether you are running Artifactory Pro or Artifactory OSS:

Running Artifactory Pro in a container

```
$ docker run --name artifactory -d -p 8081:8081  
docker.bintray.io/jfrog/artifactory-pro:latest
```

or

Running Artifactory OSS in a container

```
$ docker run --name artifactory -d -p 8081:8081  
docker.bintray.io/jfrog/artifactory-oss:latest
```

Managing Data Persistence

For your data and configuration to remain once the Artifactory Docker container is removed, you need to store them on an external volume

mounted to the Docker container. There are two ways to do this:

- Using Host Directories
- Using a Docker Named Volume

Using Host Directories

The external volume is a directory in your host's file system (such as `/var/opt/jfrog/artifactory`). When you pass this to the `docker run` command, the Artifactory process will use it to read configuration and store its data.

To mount the above example, you would use the following command:

```
$ docker run --name artifactory-pro -d -v
/var/opt/jfrog/artifactory:/var/opt/jfrog/artifactory -p 8081:8081
docker.bintray.io/jfrog/artifactory-pro:latest
```

This mounts the `/var/opt/jfrog/artifactory` directory on your host machine to the container's `/var/opt/jfrog/artifactory` and will then be used by Artifactory for configuration and data.

Using a Docker Named Volume

In this case, you create a docker named volume and pass it to the container. By default, the named volume is a local directory under `/var/lib/docker/volumes/<name>`, but can be set to work with other locations. For more details, please refer to the Docker documentation for [Docker Volumes](#).

The example below creates a Docker named volume called `artifactory_data` and mounts it to the Artifactory container under `/var/opt/jfrog/artifactory`:

```
$ docker volume create --name artifactory5_data
$ docker run --name artifactory-pro -d -v
artifactory5_data:/var/opt/jfrog/artifactory -p 8081:8081
docker.bintray.io/jfrog/artifactory-pro:latest
```

In this case, even if the container is stopped and removed, the volume persists and can be attached to a new running container using the above `docker run` command.

Upgrading Artifactory

For details on how to upgrade Artifactory running in a Docker container, please refer to [Running in a Docker Container](#) in the [Upgrading Artifactory](#) page.

Running Artifactory With a Different Database

By default, Artifactory runs with an embedded Derby Database that comes built-in, however, Artifactory supports additional databases. To switch to one of the other supported databases, please refer to [Changing the Database](#).

Building Artifactory OSS From Sources

The Artifactory OSS Docker image sources are available for download allowing you to build the image yourself. For details, please refer to [Building Artifactory OSS](#).

Accessing Artifactory

Once the Artifactory container is up and running, you access Artifactory in the usual way by browsing to:

```
http://SERVER_DOMAIN:8081/artifactory
```

For example, if you are testing on your local machine you would use: <http://localhost:8081/artifactory>

Troubleshooting Docker

This section describes different ways you can troubleshoot a running or stopped Docker container that is not functioning as expected.

Container State

The `docker ps` command lists containers in your system.

```
$ docker ps      # Lists running containers
$ docker ps -a   # Lists all containers
```

Logs

Artifactory logs are stored in the Artifactory container under `/var/opt/jfrog/artifactory/logs`.

If you ran the container with a mounted volume for Artifactory data (`/var/opt/jfrog/artifactory/`), you can also access the logs locally on your host.

An easy way to see the logged output of a running container is through the `docker logs` command

```
$ docker logs <container name>
```

This will output all of the container's STDOUT and STDERR to the screen both for running and stopped containers.

Connect to a Running Container

You can connect to a running container's file system and open an interactive command prompt in the container with the `docker exec` command

```
$ docker exec -it <container name> /bin/bash
```

This will open a command prompt in the running Artifactory container, logging you in as root and placing you in the `/` directory.

Run an Alternate Entrypoint

There are cases where you want to run the container, but not start up Artifactory. To do this, you need to override the configured entrypoint script using `docker run --entrypoint=bash`

```
$ docker run -it --entrypoint=/bin/bash -v
/var/opt/jfrog/artifactory:/var/opt/jfrog/artifactory -p 8081:8018
docker.bintray.io/jfrog/artifactory-pro:latest
```

This will run the container, presenting you with a prompt in the container, but without executing the `/entrypoint-artifactory.sh` file.

You can then make changes to the container configuration execute `/entrypoint-artifactory.sh` to start up Artifactory in your

container.

Watch the Screencast

Building Artifactory OSS

Overview

The Artifactory OSS Docker image sources are freely available for download allowing you to build and if necessary, tune it according to your needs.

Getting the Artifactory OSS Sources

For Artifactory OSS, the latest sources can be downloaded from Bintray to be built on your own machines using the following link:

Page contents

- Overview
- Getting the Artifactory OSS Sources
- Building the Artifactory OSS Docker Image
- Running the Artifactory OSS Container
- Accessing Artifactory

Building the Artifactory OSS Docker Image

To build the image, you can use the Docker native client or install Docker for your platform by downloading [Docker for Mac](#) or [Docker for Windows](#).

The following code snippet shows an example of how to build the Artifactory OSS 5.0 Docker image on a Mac:

```
$ mkdir ~/git/artifactory.5.0/
$ cd ~/git/artifactory.5.0/

$ # unzip the Artifactory source bundle you downloaded from Bintray in
to the directory you created
$ cd ~/git/artifactory.5.0/artifactory-oss/distribution/docker
$ mvn clean package -Pdocker
```

Running the Artifactory OSS Container

You can list the Docker images you have built using the **docker images** command, which should display something like the following output:

```
$ docker images
REPOSITORY              TAG                IMAGE ID
CREATED                SIZE
jfrog/artifactory-oss   5.0.0             da70b82904e7     2
minutes ago           741.7 MB
...
```

To start an Artifactory OSS container for the image shown in the example above, use:

```
$ docker run --name artifactory-5.0.0 -d -p 8081:8081
jfrog/artifactory-oss:5.0.0
```

Accessing Artifactory

Once the Artifactory container is up and running, you access Artifactory in the usual way by browsing to:

http://SERVER_DOMAIN:8081/artifactory.

For example, if you are testing on your local machine you would use: <http://localhost:8081/artifactory>

Changing the Database

Overview

By default, Artifactory runs with an embedded Derby database that comes built-in. However, Artifactory supports additional databases as described in [Configuring the Database](#).

To configure your Artifactory Docker container to run with one of the other supported databases, you need to:

1. Mount the relevant database driver into Artifactory's `tomcat/lib` directory

Using PostgreSQL

If you are changing the database to PostgreSQL, note that the Artifactory Docker image comes pre-loaded with the PostgreSQL database driver.

2. Pass environment variables to Artifactory in the `docker run` command
3. Pass database parameters as Docker environment variables telling Artifactory how to configure the database in the `docker run` command

Page contents

- Overview
- Mounting the Database Driver
- Passing Environment Variables
- Pass Database Parameters
- Examples
 - PostgreSQL
 - MySQL

Mounting the Database Driver

To mount the database driver, you first need to download its corresponding jar file from the vendor's web site provided in the following links:

- [MySQL](#)
- [Oracle JDBC](#)
- [MS SQL](#)
- [PostgreSQL](#)

Passing Environment Variables

Once you have the downloaded the database driver JAR file, you mount it into the Artifactory container using `docker run` with the `-v` option:

```
docker run ... -v
</path/to/driver.jar>:/opt/jfrog/artifactory/tomcat/lib/<driver.jar>
```

Without passing the DB parameters mentioned below, Artifactory will ignore the added jar.

Pass Database Parameters

For Artifactory in Docker to know what database to use, you need to pass in some parameters as Docker environment variables using `docker run` with the `-e` option

```
docker run ... -e PARAM=<value>
```

The following table describes the parameters supported:

DB_TYPE	Values: postgresql , mysql , oracle or mssql Default: blank indicating Artifactory should run with the built-in Derby database
DB_HOST	The hostname/ip of the server where the database is installed If this value is omitted, DB_HOST defaults to the value set in DB_TYPE
DB_PORT	The database port Defaults to the value set in the corresponding <i>db.properties</i> file
DB_URL	The full database URL Defaults to the value set in the corresponding <i>db.properties</i> file
DB_USER	The database username Defaults to the value set in the corresponding <i>db.properties</i> file
DB_PASSWORD	The database password Defaults to the value set in the corresponding <i>db.properties</i> file

During execution of the `docker run`, an entrypoint script will copy the matching *db.properties* from the *misc/db/* directory to the *etc/* directory and configure it.

If *etc/db.properties* already exists, the script will just validate it and Artifactory will work with the specified database.

Examples

Below are examples that show how Artifactory can be run in Docker with a custom database.

PostgreSQL

In this example for PostgreSQL, since only DB_TYPE is specified, the rest of the parameters will be set to their defaults taken from the PostgreSQL *db.properties* file.

The PostgreSQL database driver is already in the Artifactory Docker image, so you don't need to mount it.

```
$ docker run -d --name artifactory-5 \  
    -e DB_TYPE=postgresql \  
    -v /var/opt/jfrog/artifactory:/var/opt/jfrog/artifactory \  
    -p 8081:8081 docker.bintray.io/jfrog/artifactory-pro:latest
```

You can verify that Artifactory is running with PostgreSQL under **Admin | System Info** which specifies the **Database Type**.

MySQL

This example for MySQL uses custom database host, port, username and password settings.

The MySQL database driver is mounted in to the container's */opt/jfrog/artifactory/tomcat/lib/* directory.

```
$ docker run -d --name artifactory-5 \  
  -e DB_TYPE=mysql \  
  -e DB_HOST=mysql5srv.jfrog.local \  
  -e DB_PORT=33307 \  
  -e DB_USER=artifactory17 \  
  -e DB_PASSWORD=pass17arti56_x \  
  -v \  
~/mysql-connector-java-5.1.40-bin.jar:/opt/jfrog/artifactory/tomcat/lib/  
mysql-connector-java-5.1.40-bin.jar \  
  -v /var/opt/jfrog/artifactory:/var/opt/jfrog/artifactory \  
  -p 8081:8081 docker.bintray.io/jfrog/artifactory-pro:latest
```

You can verify that Artifactory is running with MySQL under **Admin | System Info** which specifies the **Database Type**.

Upgrading Artifactory

Overview

The procedure to upgrade Artifactory depends on your installation type. We strongly recommend reading through this page before proceeding with your upgrade. Detailed upgrade instructions are provided in dedicated pages for the following installation types:

- ZIP file
- RPM
- Debian
- Docker

In addition, within each installation type, there may be slight variation in instructions if you are upgrading from older versions of Artifactory.

Before You Proceed

Before you upgrade

We strongly recommend that you take the following actions to ensure you can roll back your system in case you encounter any issues during the upgrade process:

1. Do a complete [System Export](#) before commencing your upgrade procedure. If at any time you decide to roll back to your current version, you can use the export to reproduce your current system in its entirety.
2. Back up your database.

Before proceeding, there are a few points you need to address:

1. JDK Version

From version 4.0, Artifactory requires JDK 8. If your current version is v3.x, before you upgrade to Artifactory 5.x, please make sure you install JDK 8 and update your `JAVA_HOME` environment variable to point to your JDK 8 installation. For more details, please refer to [System Requirements](#).

2. Repositories with Multiple Package Types

From version 4.0, Artifactory will only index, and work with corresponding clients for single package type repositories. If your current version is 3.x and the installation includes repositories that support multiple package types, you need to migrate them to single package type repositories. You may do so before upgrading or after. For more details please refer to [Single Package Type Repositories](#).

3. 'Slash' character encoding for NPM builds

Handling of 'slash' character encoding for NPM has been moved from the `artifactory.system.properties` file to the `atalina.properties` file of your Tomcat. For details, please refer to [Npm Scope Packages](#).

Page Contents

- Overview
 - Before You Proceed
- Upgrading Artifactory Enterprise / HA
- Upgrading to the Latest Version
 - ZIP Installation
 - Debian Installation
 - Docker Installation
 - RPM Installation
 - RPM OSS Installation
- Using SHA256 Checksums
- Upgrading from OSS to Pro
- Upgrading from Version 3.x
 - Single Package Type Repositories
 - Migrating to Single Package Type Repositories
 - Fixing Multiple Package Type Repositories
- Upgrading from Any Version Below v3.0
- Downgrading Artifactory
- Watch the Screencast

Learn more

- [Upgrading an Enterprise HA Cluster](#)

Upgrading Artifactory Enterprise / HA

There are different instructions for upgrading Artifactory HA

When upgrading an HA cluster, the procedure for upgrading each node is similar to upgrading a single instance (Non-HA) installation, however, there are additional actions required for the nodes to operate as a high availability cluster.

If you are upgrading an Artifactory HA cluster, please refer to [Upgrading an Enterprise HA Cluster](#).

Upgrading to the Latest Version

Upgrading from version 4.x or 5.x to the latest version is a simple procedure. Please refer to the sections below with specific instructions for your installation type.

ZIP Installation

▼ Upgrading a ZIP installation...

1. Unzip the Artifactory distribution archive.
2. If the `ARTIFACTORY_HOME/tomcat/conf/server.xml` has been modified keep it in a temporary location.
3. Backup files to a temporary location according to the conditions described below:
 - a. In all cases, backup `ARTIFACTORY_HOME/bin/artifactory.default`
 - b. If Artifactory is configured to work with a database that is not Derby, backup the `ARTIFACTORY_HOME/tomcat/lib/<JDBC>` driver.
4. Remove the following files and folders from your `ARTIFACTORY_HOME` folder:
 - `webapps/artifactory.war`
 - `webapps/access.war` (this will only be present if your current version is 5.4.0 and above due to addition of the Access Service)
 - `tomcat`
 - `bin`
 - `misc`
5. Replace the removed files and folders with the corresponding ones from the new unzipped version.
6. Any files that were stored in temporary locations should now be returned to their original location under the new installation.

Running Artifactory as ROOT

If you have configured Artifactory to run as the ROOT application in Tomcat, before you proceed, you need to follow the steps described in [this Knowledge Base article](#).

Upgrading to version 5.4.0 and above

From version 5.4.0, Artifactory's management of Access Tokens was moved to a separate service, Access, installed as a separate web application under the same Tomcat as Artifactory. This requires your Tomcat's `server.xml` to be configured to allow running 2 processes. If you are using a `server.xml` file from a previous installation, when returning it, make sure it is configured to allow 2 start/stop threads as shown below (see `<Host name="localhost" appBase="webapps" startStopThreads="2"/>`):

```
...
    <Engine name="Catalina" defaultHost="localhost">
        <Host name="localhost" appBase="webapps"
startStopThreads="2" />
    </Engine>
...
```

7. If you installed Artifactory as a service, you now need to run the service

- For a Linux service, browse to `$ARTIFACTORY_HOME/bin` and execute the following command **as root**: `$ARTIFACTORY_HOME/bin/installService.sh [USER [GROUP]]`
- For Windows service, browse to `%ARTIFACTORY_HOME%\bin` and run `InstallService.bat`.

Debian Installation

▼ Upgrading a Debian installation...

- Log in as root (or use `sudo su -`).
- Execute the following command:

```
dpkg -i $jfrog-artifactory-<oss|pro>-5.y.z.deb
```

Managing Configuration Files

When upgrading a **Debian** installation the upgrade process overwrites the following set of configuration files:

- `system.properties`
- `config.xml`
- `default`
- `logback.xml`
- `mimetypes.xml`
- All files under `opt/jfrog/artifactory/misc`
- All files under `opt/jfrog/artifactory/webapps`

If any of these files were modified, a backed up file will be created automatically with a notification in the upgrade log. If you need to restore the configuration changes you can restore them from the backup created.

Running Artifactory as ROOT

If you have configured Artifactory to run as the ROOT application in Tomcat, before you proceed, you need to follow the steps described in [this Knowledge Base article](#).

Upgrading to version 5.4.0 and above

From version 5.4.0, Artifactory's management of Access Tokens was moved to a separate service, Access, installed as a separate web application under the same Tomcat as Artifactory. This requires your Tomcat's `server.xml` to be configured to allow running 2 processes. If you are using a `server.xml` file from a previous installation, when returning it, make sure it is configured to allow 2 start/stop threads as shown below (see `<Host name="localhost" appBase="webapps" startStopThreads="2"/>`):

```
...
    <Engine name="Catalina" defaultHost="localhost">
        <Host name="localhost" appBase="webapps"
startStopThreads="2" />
    </Engine>
...
```

Docker Installation

▼ Upgrading a Docker installation...

In order to keep your data and configuration between versions, when upgrading the Artifactory Docker image, you need to use an external mounted volume as described under [Managing Data Persistence](#).

To upgrade the Artifactory Docker image, follow these steps:

1. Stop current container
2. Start a container with new version, using same data and configuration
3. Remove old container

The example below shows this process for upgrading Artifactory from v5.0.0 to v5.1.0.

```
$ # Stop the currently running container
$ docker stop artifactory-5.0.0

$ # Start a new container from the new version image
$ docker run -d --name artifactory-5.1.0
--volumes-from=artifactory-5.0.0 -p 8081:8081
docker.bintray.io/jfrog/artifactory-pro:5.1.0

$ # Remove old container
$ docker rm artifactory-5.0.0
```

Once these commands have completed successfully, you would have the new version (5.1.0 in the above example) running with the data and configuration from the old version that was removed.

Running Artifactory as ROOT

If you have configured Artifactory to run as the ROOT application in Tomcat, you need to follow the steps described in [this Knowledge Base article](#).

Upgrading to version 5.4.0 and above

From version 5.4.0, Artifactory's management of Access Tokens was moved to a separate service, Access, installed as a separate web application under the same Tomcat as Artifactory. This requires your Tomcat's server.xml to be configured to allow running 2 processes. If you are using a server.xml file from a previous installation, when returning it, make sure it is configured to allow 2 start/stop threads as shown below (see <Host name="localhost" appBase="webapps" **startStopThreads="2"/>**):

```
...
    <Engine name="Catalina" defaultHost="localhost">
        <Host name="localhost" appBase="webapps"
startStopThreads="2" />
    </Engine>
...
```

RPM Installation

▼ Upgrading an RPM installation...

Make sure you are upgrading from v3.6 or above

When running as an RPM installation, you can only upgrade to v5.x if your current version is 3.6 or above. If necessary, first upgrade your current version to 3.6, and then upgrade to v5.x .

If you try to upgrade a version below 3.6 using `rpm --force` you may end up deleting all of your data.

1. Download the **Artifactory Pro RPM Installer**. The latest version can be downloaded from the [JFrog Artifactory Pro Download Page](#). Previous versions can be downloaded from [JFrog Bintray](#).
2. Log in as root (or use `sudo su -`).
3. Execute the following command:

```
rpm -U jfrog-artifactory-pro-5.y.z.rpm
```

Switching from Artifactory OSS to Pro

If you are just switching from Artifactory OSS to Pro with the same version number, you need to append the command with `--force --nodeps` as follows:

```
rpm -U jfrog-artifactory-pro-5.y.z.rpm --force --nodeps
```

During an upgrade of an RPM installation different files may get backed up, where the backup file is appended with either a `.rpmorig` or a `.rpmnew` extension.

A `.rpmorig` extension means that the original file in your installation, the one that was there before performing the upgrade, was backed up before being replaced in the upgrade process.

A `.rpmnew` extension means that the original file in your installation, was **not** replaced in the upgrade, and instead, the new file with the same filename was backed up.

In either case, Artifactory will display a message such as:

```
warning: /etc/opt/jfrog/artifactory/default saved as /etc/opt/jfrog/artifactory/default.rpmorig
```

In these cases we recommend comparing the file installed once the upgrade has been completed with the backed-up file to see which best fits your needs, and using that one in the final setup.

If you make any changes, you may need to restart Artifactory for the change to be applied.

Upgrading Using YUM

An easy way to upgrade Artifactory from version 3.x or 4.x to the latest version is to use YUM with the Bintray Artifactory repository. The code snippets below show how to do this depending on whether your current version is below 3.6, or 3.6 and above.

Upgrading an Artifactory HA cluster?

If you are upgrading an Artifactory HA cluster, and you are running with a version that is older than version **5.4.6**, you should review the instructions on [Upgrading an Enterprise HA Cluster](#) prior to upgrading.

▼ If your current version is 3.6 and above:

```
curl https://bintray.com/jfrog/artifactory-pro-rpms/rpm -o
bintray-jfrog-artifactory-pro-rpms.repo && sudo mv
bintray-jfrog-artifactory-pro-rpms.repo /etc/yum.repos.d
yum install jfrog-artifactory-pro
```

▼ If your current version is below 3.6:

```
curl https://bintray.com/jfrog/artifactory-pro-rpms/rpm -o
bintray-jfrog-artifactory-pro-rpms.repo && sudo mv
bintray-jfrog-artifactory-pro-rpms.repo /etc/yum.repos.d
yum upgrade artifactory
yum install jfrog-artifactory-pro
```

Running Artifactory as ROOT

If you have configured Artifactory to run as the ROOT application in Tomcat, before you proceed, you need to follow the steps described in [this Knowledge Base article](#).

Upgrading to version 5.4.0 and above

From version 5.4.0, Artifactory's management of Access Tokens was moved to a separate service, Access, installed as a separate web application under the same Tomcat as Artifactory. This requires your Tomcat's `server.xml` to be configured to allow running 2 processes. If you are using a `server.xml` file from a previous installation, when returning it, make sure it is configured to allow 2 start/stop threads as shown below (see `<Host name="localhost" appBase="webapps" startStopThreads="2"/>`):

```
...
    <Engine name="Catalina" defaultHost="localhost">
        <Host name="localhost" appBase="webapps"
startStopThreads="2" />
    </Engine>
...
```

RPM OSS Installation

▼ [Click here to expand...](#)

1. Download the **Artifactory OSS RPM Installer**. The latest version can be downloaded from the [JFrog Open Source](#) page. Previous versions can be downloaded from [JFrog Bintray](#).
2. Log in as root (or use `sudo su -`).
3. Execute the following command:

```
rpm -U jfrog-artifactory-oss-5.y.z.rpm
```

During an upgrade of an RPM installation different files may get backed up, where the backup file is appended with either a `.rpmorig` or a `.rpmnew` extension.

A `.rpmorig` extension means that the original file in your installation, the one that was there before performing the upgrade, was backed up before being replaced in the upgrade process.

A `.rpmnew` extension means that the original file in your installation, was **not** replaced in the upgrade, and instead, the new file with the same filename was backed up.

In either case, Artifactory will display a message such as:

```
warning: /etc/opt/jfrog/artifactory/default saved as /etc/opt/jfrog/artifactory/default.rpmorig
```

In these cases we recommend comparing the file installed once the upgrade has been completed with the backed-up file to see which best fits your needs, and using that one in the final setup.

If you make any changes, you may need to restart Artifactory for the change to be applied.

Upgrading Using YUM

An easy way to upgrade Artifactory from version 3.x or 4.x to the latest version is to use YUM with the Bintray Artifactory repository. The code snippets below show how to do this depending on whether your current version is below 3.6, or 3.6 and above.

▼ [If your current version is 3.6 and above:](#)

```
curl https://bintray.com/jfrog/artifactory-rpms/rpm -o
bintray-jfrog-artifactory-rpms.repo && sudo mv
bintray-jfrog-artifactory-rpms.repo /etc/yum.repos.d/
yum install jfrog-artifactory-oss
```

▼ [If your current version is below 3.6:](#)

```
curl https://bintray.com/jfrog/artifactory-rpms/rpm -o
bintray-jfrog-artifactory-rpms.repo && sudo mv
bintray-jfrog-artifactory-rpms.repo /etc/yum.repos.d/
yum upgrade artifactory
yum install jfrog-artifactory-oss
```

Running Artifactory as ROOT

If you have configured Artifactory to run as the ROOT application in Tomcat, before you proceed, you need to follow the steps described in [this Knowledge Base article](#).

Upgrading to version 5.4.0 and above

From version 5.4.0, Artifactory's management of Access Tokens was moved to a separate service, Access, installed as a separate web application under the same Tomcat as Artifactory. This requires your Tomcat's server.xml to be configured to allow running 2 processes. If you are using a server.xml file from a previous installation, when returning it, make sure it is configured to allow 2 start/stop threads as shown below (see <Host name="localhost" appBase="webapps" **startStopThreads="2"/>):**

```
...
    <Engine name="Catalina" defaultHost="localhost" >
        <Host name="localhost" appBase="webapps"
startStopThreads="2" />
    </Engine>
...
```

Using SHA256 Checksums

From version 5.5, Artifactory natively supports SHA-256. New artifacts that are uploaded will automatically have their SHA-256 checksum calculated, however, artifacts that were already hosted in Artifactory prior to the upgrade will not have their SHA-256 checksum in the database yet.

To make full use of Artifactory's SHA-256 capabilities after upgrading to version 5.5 and above, you need to run a process that migrates Artifactory's database making sure that the record for each artifact includes its SHA-256 checksum. For full details on Artifactory's SHA-256 support and instructions on how to migrate your database, please refer to [SHA-256 Support](#).

Upgrading from OSS to Pro

Even if you're just switching your current version of Artifactory OSS to the same version of Artifactory Pro, please follow the instructions under [Upgrading to the Latest Version](#) according to your installation type (ZIP, RPM, Debian or Docker)

Upgrading from Version 3.x

Single Package Type Repositories

Single Package Type

To work with version 4.x or 5.x, you need to ensure that your repositories only contain artifacts with the same package type. A script to check this can be found on the [JFrog GitHub](#).

In version 3.x Artifactory supported repositories with multiple package types. You were able to upload packages with different types to the same repository and Artifactory would calculate the metadata for those packages. Nevertheless, maintaining a single package type per repository was always a best practice that optimized performance and produced a more organized repository structure in your system. From version 4.0, you need to specify a single **Package Type** for a repository when you create it. Artifactory will only calculate metadata for, and be

recognized by the corresponding client software for artifacts of the **Package Type** specified for that repository. (Artifactory will not prevent you from uploading packages of a different type, however, it will not calculate metadata for those packages, and the client for the different package types will not recognize the repository).

If you currently have repositories that are configured to support multiple package types, you need to migrate them to single package type repositories, however, you may do so either before or after running the upgrade procedure.

To migrate your repositories before upgrading, please refer to [Migrating to Single Package Type Repositories](#).

If you prefer to migrate your repositories after upgrading, or have already upgraded, please refer to [Fixing Multiple Package Type Repositories](#).

Generic repositories

In version 4.x and 5.x, if you need a repository to hold packages of several different types, you may specify its package type to be **Generic**. Artifactory does not calculate metadata for Generic repositories, and effectively, they behave like a simple file system to store packages.

Migrating to Single Package Type Repositories

To migrate a repository with multiple package types to single package type repositories, execute the following steps:

1. Change the configuration of the original repository so it supports only one package type.
2. For each additional packaging type needed, create a new repository with the corresponding package type
3. Use the REST API or the UI to move packages from the original repository to the new one(s) created until all repositories only contain packages of the same type.
When using the REST API, make sure to include the `suppressLayouts=1` query parameter in order to prevent artifact path transformations.

Npm Repositories

If you move data to an Npm repository, make sure to include the `.npm` folder. This will preserve extra information that may have been stored when deploying packages using the npm client.

Fixing Multiple Package Type Repositories

If you upgraded without migrating to single package type repositories, then Artifactory will start normally, however, repositories containing multiple package types will be **randomly** assigned one of the single package types from the original repository and output corresponding messages to the `$ARTIFACTORY_HOME/logs/artifactory.log` file.

For example, if `libs-release-local` contained three different package types: RubyGems, Npm and NuGet, after upgrading, your `$ARTIFACTORY_HOME/logs/artifactory.log` may contain messages similar to the ones below:

```
2015-06-28 10:10:47,656 [art-init] [INFO ]
(o.a.v.c.v.SingleRepoTypeConverter:42) Converting repositories to a
single package type
2015-06-28 10:10:47,663 [art-init] [ERROR]
(o.a.v.c.v.SingleRepoTypeConverter:155) Disabling package 'Gems' for
repo 'libs-release-local' since only one packaging type is allowed!
2015-06-28 10:10:47,664 [art-init] [ERROR]
(o.a.v.c.v.SingleRepoTypeConverter:155) Disabling package 'Npm' for repo
'libs-release-local' since only one packaging type is allowed!
2015-06-28 10:10:47,664 [art-init] [INFO ]
(o.a.v.c.v.SingleRepoTypeConverter:128) Setting repository
'libs-release-local' to type NuGet
2015-06-28 10:10:47,664 [art-init] [INFO ]
(o.a.v.c.v.SingleRepoTypeConverter:128) Setting repository
'libs-snapshot-local' to type Maven
2015-06-28 10:10:47,664 [art-init] [INFO ]
(o.a.v.c.v.SingleRepoTypeConverter:128) Setting repository
'plugins-release-local' to type Maven
2015-06-28 10:10:47,664 [art-init] [INFO ]
(o.a.v.c.v.SingleRepoTypeConverter:128) Setting repository
'plugins-snapshot-local' to type Maven
2015-06-28 10:10:47,665 [art-init] [INFO ]
(o.a.v.c.v.SingleRepoTypeConverter:128) Setting repository
'ext-release-local' to type Maven
2015-06-28 10:10:47,665 [art-init] [INFO ]
(o.a.v.c.v.SingleRepoTypeConverter:128) Setting repository
'ext-snapshot-local' to type Maven
2015-06-28 10:10:47,666 [art-init] [INFO ]
(o.a.v.c.v.SingleRepoTypeConverter:128) Setting repository 'jcenter' to
type Maven
2015-06-28 10:10:47,666 [art-init] [INFO ]
(o.a.v.c.v.SingleRepoTypeConverter:128) Setting repository 'remote-repo'
to type Maven
2015-06-28 10:10:47,668 [art-init] [INFO ]
(o.a.v.c.v.SingleRepoTypeConverter:128) Setting repository
'libs-snapshot' to type Maven
2015-06-28 10:10:47,668 [art-init] [INFO ]
(o.a.v.c.v.SingleRepoTypeConverter:128) Setting repository 'p2' to type
P2
2015-06-28 10:10:47,668 [art-init] [INFO ]
(o.a.v.c.v.SingleRepoTypeConverter:56) Finished Converting repositories
to a single package type
```

In this example, Artifactory set the **Package Type** to NuGet.

To fix this condition, you can simply follow steps described above in [Migrating to Single Package Type Repositories](#), or after you upgrade, use the `packageType` utility found on the [JFrog Github](#) for 4.x migrations.

Upgrading from Any Version Below v3.0

To upgrade from a version prior to 3.0, you first need to upgrade to version 3.9.x as described in [Upgrading Artifactory in the Artifactory 3 documentation](#).

Interim versions

Depending on your current version, upgrading to version 3.9.x may require you to first upgrade to an interim version.

Downgrading Artifactory

The procedure to downgrade Artifactory may vary depending on the version you are using. For more details, please contact [JFrog Support](#).

Watch the Screencast

Upgrading an Enterprise HA Cluster

Overview

This page describes the process to upgrade your Artifactory Enterprise HA cluster.

No down time

Since your cluster contains more than one node, you may complete the upgrade process without incurring any down time to the Artifactory service your organization is using.

Upgrading from 5.4.5 and below to 5.5 and above

In version 5.5, Artifactory's database underwent a schema change to accommodate SHA256 checksums. As a result, when upgrading from version 5.4.5 and below to version 5.5 and above, you need to follow one of the options described in the [detailed instructions](#) below.

Before You Begin

1. **Backup your system:** As a precaution, before you begin the upgrade process, we strongly recommend performing a complete [Complete System Backup](#).
2. **Backup your database**
3. **Read through the process:** The backup procedure may vary slightly depending on your current version and your installation type (ZIP, RPM, Debian or Docker). To familiarize yourself with the specific backup process that you should use, we recommend reading through all the steps of the process before you begin.

Page Contents

- Overview
- The Upgrade Process
 - Upgrading From Version 5.4.5 and Below to Version 5.5 and Above
 - U
 - Recovering from an Attempt to Upgrade Directly to Version 5.5 and Above
 - R
 - Upgrading from Version 4.x or 5.x
 - U
 - Docker Installation
 - U
 - Docker Installation
 - V
- Troubleshooting the Upgrade Process

The Upgrade Process

Upgrading Artifactory HA depends on which version you are starting from. Read the sections below carefully to make sure you complete the process correctly.

Upgrading to 5.x for the first time? NFS is no longer required

From version 5.0, Artifactory HA does not require a network file system (NFS) to store data. If you wish to migrate your filestore to alternative storage locations, please refer to [Migrating Data from NFS](#).

Upgrading From Version 5.4.5 and Below to Version 5.5 and Above

Artifactory 5.5 implements a database schema change to support SHA-256 checksums. **If your current version is 5.4.6**, you may proceed with the normal upgrade procedure described in [Upgrading from Version 4.x or 5.x](#) below.

If your current version is below 5.4.6, to accommodate this change, you may select one of the following two upgrade options:

1. **Two-phase, zero downtime**

In this option, you first need to upgrade your HA cluster to version 5.4.6. Once this upgrade is completed, you can then proceed to upgrade your HA cluster to version 5.5. In both phases, you follow the normal upgrade procedure described in [Upgrading from Version 4.x or 5.x](#) below.

2. **One phase with downtime**

This option requires you to execute the following preprocessing

- While the primary and all secondary nodes are up and running, add the following flag to `artifactory.system.properties` on the **primary node**:

```
artifactory.upgrade.allowAnyUpgrade.forVersion=<to_version>
For example:
artifactory.upgrade.allowAnyUpgrade.forVersion=5.5.0
```

- Wait until this property is synchronized to the database. You can verify that it has been synchronized by checking the `artifactory.log` file in each of the secondary nodes. Your nodes should display a message that looks like this:

```
[Node ID: some_node_id] detected remote modify for config 'artifactory.system.properties'
```

- Now you can proceed with the normal upgrade procedure described in [Upgrading from Version 4.x or 5.x](#) below.

System will be down during this particular upgrade

Normally, upgrading an enterprise HA cluster does not incur downtime. As you upgrade each node, the other nodes continue to provide service.

In this particular scenario of upgrading from version 5.4.5 and below to version 5.5 and above in a single phase, once you upgrade your primary node, your cluster will be in an incoherent state and will not be able to provide service until all nodes in the cluster have been upgraded. You can expect to see many errors in the log file until the upgrade is complete.

As a result, we recommend completely taking down the whole cluster (i.e. taking down all cluster nodes), and then following the upgrade procedure, bringing up each node as it is upgraded.

If you try upgrading directly to version 5.5 **without** following one of these options, the upgrade will fail and the following message will be logged in the `artifactory.log` file:

```
To upgrade your HA installation to this version, you first need to upgrade to version 5.4.6 which implements changes required to accommodate a database schema change.
```

Using SHA256 Checksums

From version 5.5, Artifactory natively supports SHA-256. New artifacts that are uploaded will automatically have their SHA-256 checksum calculated, however, artifacts that were already hosted in Artifactory prior to the upgrade will not have their SHA-256 checksum in the database yet.

To make full use of Artifactory's SHA-256 capabilities after upgrading to version 5.5 and above, you need to run a process that migrates Artifactory's database making sure that the record for each artifact includes its SHA-256 checksum. For full details on Artifactory's SHA-256 support and instructions on how to migrate your database, please refer to [SHA-256 Support](#).

Recovering from an Attempt to Upgrade Directly to Version 5.5 and Above

If you try to upgrade from version 5.4.5 and below to version 5.5 and above without following the procedure in one of the two options above, the upgrade will fail with the following message:

```
To upgrade your HA installation to this version, you first need to upgrade to version 5.4.6 which implements changes required to accommodate a database schema change.
```

Depending on your installation type, the message may be displayed in the `artifactory.log` file, the `$ARTIFACTORY_HOME/tomcat/log/s/localhost.log` file or in the command line output.

To recover from this error and, you first need to replace your current version as described [below](#) (according to your installation type).

Then you can proceed with upgrading to version 5.4.6 as required using the [normal upgrade procedure](#), and then on to your final desired version, also, using the [normal upgrade procedure](#).

Reinstalling Your Current Version

Reinstalling is similar to the process you went through when upgrading to your current version according to your installation type.

For example, if you tried to upgrade from version 5.2 directly to version 5.5, you now need to reinstall version 5.2 and follow the instructions below, according to your installation type.

Zip Installation

Follow the [upgrade instructions for a Zip installation](#) using your current version.

Debian Installation

Follow the [upgrade instructions for a Debian installation](#) using your current version.

If your current version is below 5.4.0, make sure to remove the `$ARTIFACTORY_HOME/tomcat/webapps/access` folder. This folder is a remnant of the failed attempt to upgrade to version 5.5 and is not needed in versions previous to 5.4 where the [Access Service](#) is bundled together with the Artifactory WAR.

Docker Installation

Follow the [upgrade instructions for a Docker installation](#) using your current version.

RPM Installation

Follow the [upgrade instructions for an RPM installation](#) using your current version.

If your current version is below 5.4.0, make sure to remove the `$ARTIFACTORY_HOME/tomcat/webapps/access` folder. This folder is a remnant of the failed attempt to upgrade to version 5.5 and is not needed in versions previous to 5.4 where the [Access Service](#) is bundled together with the Artifactory WAR.

Upgrading from Version 4.x or 5.x

Upgrading to the latest version is conducted in three phases:

1. Upgrading the primary node
2. Upgrading the secondary nodes
3. Verifying the HA installation and configuration

Want to stop using NFS?

If you want to stop using a shared NFS once the upgrade procedure is complete (**this is optional**), please refer to [Migrating Data from NFS](#) to migrate to alternative storage.

Upgrading the Primary Node

1. Remove the primary node from the load balancer, so all requests are directed to the secondary nodes. You may look at `$ARTIFACTORY_NODE_HOME/logs/request.log` and `ARTIFACTORY_URL/api/tasks` (search for "running") to ensure that Artifactory is completely inactive.
2. Perform a graceful shutdown of the primary node. While the primary node is down, the load balancer should redirect all queries to the secondary nodes.
3. Continue with the upgrade according to the instructions for your installation type.

Upgrading from 3.x?

If your current version is 3.5 or higher, you first need to upgrade to the latest version 4.x using the [procedure in this link](#) and then upgrade to 5.x.

Upgrading Artifactory HA from a version below 3.5 to version 5.x directly is not supported. To upgrade to version 5.x, you first need to upgrade your system to v3.5+, and then upgrade again to the final version 4.x, and then finally to 5.x.

ZIP Installation

▼ Upgrading a ZIP installation...

- a. Unzip the Artifactory distribution archive.
- b. If the `$ARTIFACTORY_HOME/tomcat/conf/server.xml` has been modified keep it in a temporary location.

- c. Backup files to a temporary location according to the conditions described below:
 - i. In all cases, backup `$ARTIFACTORY_HOME/bin/artifactory.default`
 - ii. If Artifactory is configured to work with a database that is not Derby, backup the `$ARTIFACTORY_HOME/tomcat/ib/<JDBC>` driver.
- d. Remove the following files and folders from your `$ARTIFACTORY_HOME` folder:
 - `webapps/artifactory.war`
 - `webapps/access.war` (this will only be present if your current version is 5.4.0 and above due to addition of the Access Service)
 - `tomcat`
 - `bin`
 - `misc`
- e. Replace the removed files and folders with the corresponding ones from the new unzipped version.
- f. Any files that were stored in temporary locations should now be returned to their original location under the new installation.

Running Artifactory as ROOT

If you have configured Artifactory to run as the ROOT application in Tomcat, before you proceed, you need to follow the steps described in [this Knowledge Base article](#).

Upgrading to version 5.4.0 and above

From version 5.4.0, Artifactory's management of Access Tokens was moved to a separate service, Access, installed as a separate web application under the same Tomcat as Artifactory. This requires your Tomcat's `server.xml` to be configured to allow running 2 processes. If you are using a `server.xml` file from a previous installation, when returning it, make sure it is configured to allow 2 start/stop threads as shown below (see `<Host name="localhost" appBase="webapps" startStopThreads="2"/>`):

```

...
    <Engine name="Catalina"
defaultHost="localhost">
        <Host name="localhost" appBase="webapps"
startStopThreads="2"/>
    </Engine>
...

```

- g. If you installed Artifactory as a service, you now need to run the service
 - i. For a Linux service, browse to `$ARTIFACTORY_HOME/bin` and execute the following command **as root**: `$ARTIFACTORY_HOME/bin/installService.sh [USER [GROUP]]`
 - ii. For Windows service, browse to `%ARTIFACTORY_HOME%\bin` and run `InstallService.bat`.

Debian Installation

▼ Upgrading a Debian installation...

- a. Log in as root (or use `sudo su -`).
- b. Execute the following command:

```
dpkg -i $jfrog-artifactory-<oss|pro>-5.y.z.deb
```

Managing Configuration Files

When upgrading a **Debian** installation the upgrade process overwrites the following set of configuration files:

- `system.properties`
- `config.xml`
- `default`
- `logback.xml`
- `mimetypes.xml`
- All files under `opt/jfrog/artifactory/misc`
- All files under `opt/jfrog/artifactory/webapps`

If any of these files were modified, a backed up file will be created automatically with a notification in the upgrade log. If you need to restore the configuration changes you can restore them from the backup created.

Running Artifactory as ROOT

If you have configured Artifactory to run as the ROOT application in Tomcat, before you proceed, you need to follow the steps described in [this Knowledge Base article](#).

Upgrading to version 5.4.0 and above

From version 5.4.0, Artifactory's management of Access Tokens was moved to a separate service, Access, installed as a separate web application under the same Tomcat as Artifactory. This requires your Tomcat's server.xml to be configured to allow running 2 processes. If you are using a server.xml file from a previous installation, when returning it, make sure it is configured to allow 2 start/stop threads as shown below (see <Host name="localhost" appBase="webapps" startStopThreads="2"/>):

```
...
    <Engine name="Catalina" defaultHost="localhost">
        <Host name="localhost" appBase="webapps"
startStopThreads="2"/>
    </Engine>
...
```

Docker Installation

▼ Upgrading a Docker installation...

In order to keep your data and configuration between versions, when upgrading the Artifactory Docker image, you need to use an external mounted volume as described under [Managing Data Persistence](#).

To upgrade the Artifactory Docker image, follow these steps:

- a. Stop current container
- b. Start a container with new version, using same data and configuration
- c. Remove old container

The example below shows this process for upgrading Artifactory from v5.0.0 to v5.1.0.

```
$ # Stop the currently running container
$ docker stop artifactory-5.0.0

$ # Start a new container from the new version image
$ docker run -d --name artifactory-5.1.0
--volumes-from=artifactory-5.0.0 -p 8081:8081
docker.bintray.io/jfrog/artifactory-pro:5.1.0

$ # Remove old container
$ docker rm artifactory-5.0.0
```

Once these commands have completed successfully, you would have the new version (5.1.0 in the above example) running with the data and configuration from the old version that was removed.

Running Artifactory as ROOT

If you have configured Artifactory to run as the ROOT application in Tomcat, you need to follow the steps described in [this Knowledge Base article](#).

Upgrading to version 5.4.0 and above

From version 5.4.0, Artifactory's management of Access Tokens was moved to a separate service, Access, installed

as a separate web application under the same Tomcat as Artifactory. This requires your Tomcat's server.xml to be configured to allow running 2 processes. If you are using a server.xml file from a previous installation, when returning it, make sure it is configured to allow 2 start/stop threads as shown below (see <Host name="localhost" appBase="webapps" **startStopThreads="2"/>):**

```
...
    <Engine name="Catalina" defaultHost="localhost">
        <Host name="localhost" appBase="webapps"
startStopThreads="2"/>
    </Engine>
...
```

RPM Installation

▼ Upgrading an RPM installation...

Make sure you are upgrading from v3.6 or above

When running as an RPM installation, you can only upgrade to v5.x if your current version is 3.6 or above. If necessary, first upgrade your current version to 3.6, and then upgrade to v5.x .

If you try to upgrade a version below 3.6 using `rpm --force` you may end up deleting all of your data.

- Download the **Artifactory Pro RPM Installer**. The latest version can be downloaded from the [JFrog Artifactory Pro Download Page](#). Previous versions can be downloaded from [JFrog Bintray](#).
- Log in as root (or use `sudo su -`).
- Execute the following command:

```
rpm -U jfrog-artifactory-pro-5.y.z.rpm
```

Switching from Artifactory OSS to Pro

If you are just switching from Artifactory OSS to Pro with the same version number, you need to append the command with `--force --nodeps` as follows:

```
rpm -U jfrog-artifactory-pro-5.y.z.rpm --force --nodeps
```

During an upgrade of an RPM installation different files may get backed up, where the backup file is appended with either a `.rpmorig` or a `.rpmnew` extension.

A `.rpmorig` extension means that the original file in your installation, the one that was there before performing the upgrade, was backed up before being replaced in the upgrade process.

A `.rpmnew` extension means that the original file in your installation, was **not** replaced in the upgrade, and instead, the new file with the same filename was backed up.

In either case, Artifactory will display a message such as:

```
warning: /etc/opt/jfrog/artifactory/default saved as /etc/opt/jfrog/artifactory/default.rpmorig
```

In these cases we recommend comparing the file installed once the upgrade has been completed with the backed-up file to see which best fits your needs, and using that one in the final setup.

If you make any changes, you may need to restart Artifactory for the change to be applied.

Upgrading Using YUM

An easy way to upgrade Artifactory from version 3.x or 4.x to the latest version is to use YUM with the Bintray Artifactory repository. The code snippets below show how to do this depending on whether your current version is below 3.6, or 3.6 and above.

Upgrading an Artifactory HA cluster?

If you are upgrading an Artifactory HA cluster, and you are running with a version that is older than version **5.4.6**, you should review the instructions on [Upgrading an Enterprise HA Cluster](#) prior to upgrading.

▼ If your current version is 3.6 and above:

```
curl https://bintray.com/jfrog/artifactory-pro-rpms/rpm -o
bintray-jfrog-artifactory-pro-rpms.repo && sudo mv
bintray-jfrog-artifactory-pro-rpms.repo /etc/yum.repos.d
yum install jfrog-artifactory-pro
```

▼ If your current version is below 3.6:

```
curl https://bintray.com/jfrog/artifactory-pro-rpms/rpm -o
bintray-jfrog-artifactory-pro-rpms.repo && sudo mv
bintray-jfrog-artifactory-pro-rpms.repo /etc/yum.repos.d
yum upgrade artifactory
yum install jfrog-artifactory-pro
```

Running Artifactory as ROOT

If you have configured Artifactory to run as the ROOT application in Tomcat, before you proceed, you need to follow the steps described in [this Knowledge Base article](#).

Upgrading to version 5.4.0 and above

From version 5.4.0, Artifactory's management of Access Tokens was moved to a separate service, Access, installed as a separate web application under the same Tomcat as Artifactory. This requires your Tomcat's server.xml to be configured to allow running 2 processes. If you are using a server.xml file from a previous installation, when returning it, make sure it is configured to allow 2 start/stop threads as shown below (see `<Host name="localhost" appBase="webapps" startStopThreads="2"/>`):

```
...
    <Engine name="Catalina" defaultHost="localhost">
        <Host name="localhost" appBase="webapps"
startStopThreads="2" />
    </Engine>
...
```

4. Start up the primary node. When the primary node starts up, it will recognize that the HA cluster nodes are not all running the same version of Artifactory, and consequently, the system will be limited to allowing uploads and downloads.

Any attempt to perform other actions such as changing the DB schema, modifying permissions, changing repository configuration and more, are strictly blocked. This limitation will continue until all the cluster nodes are once again running the same version.

Version inconsistency generates exceptions

Running the HA cluster nodes with different versions generates exceptions. These can be seen in the log files and reflect the temporary inconsistent state during the upgrade process. This is normal and should be ignored until all the cluster nodes are, once again, running the same version.

5. Put the primary node back to the load balancer.

Upgrading the Secondary Node

For **each secondary node** in your HA cluster, perform the following steps:

1. Remove the node from the load balancer, so all requests are directed to the other nodes. You may look at `$ARTIFACTORY_NODE`

- ..._HOME/logs/request.log and ARTIFACTORY_URL/api/tasks (search for "running") to ensure that Artifactory is completely inactive.
2. Perform a graceful shutdown of the node. While the node is down, the load balancer should redirect all queries to the other nodes.
 3. Continue with the upgrade according to the instructions for your installation type.

If your current version is 3.5 or higher, you first need to upgrade to the latest version 4.x using the following procedure and then upgrade to 5.x.

Upgrading Artifactory HA from a version below 3.5 to version 5.x directly is not supported. To upgrade to version 5.x, you first need to upgrade your system to v3.5+, and then upgrade again to the final version 4.x, and then finally to 5.x.

ZIP Installation

▼ Upgrading a ZIP installation...

- a. Unzip the Artifactory distribution archive.
- b. If the `ARTIFACTORY_HOME/tomcat/conf/server.xml` has been modified keep it in a temporary location.
- c. Backup files to a temporary location according to the conditions described below:
 - i. In all cases, backup `ARTIFACTORY_HOME/bin/artifactory.default`
 - ii. If Artifactory is configured to work with a database that is not Derby, backup the `ARTIFACTORY_HOME/tomcat/lib/<JDBC> driver`.
- d. Remove the following files and folders from your `ARTIFACTORY_HOME` folder:
 - `webapps/artifactory.war`
 - `webapps/access.war` (this will only be present if your current version is 5.4.0 and above due to addition of the [Access Service](#))
 - `tomcat`
 - `bin`
 - `misc`
- e. Replace the removed files and folders with the corresponding ones from the new unzipped version.
- f. Any files that were stored in temporary locations should now be returned to their original location under the new installation.

Running Artifactory as ROOT

If you have configured Artifactory to run as the ROOT application in Tomcat, before you proceed, you need to follow the steps described in [this Knowledge Base article](#).

Upgrading to version 5.4.0 and above

From version 5.4.0, Artifactory's management of Access Tokens was moved to a separate service, Access, installed as a separate web application under the same Tomcat as Artifactory. This requires your Tomcat's `server.xml` to be configured to allow running 2 processes. If you are using a `server.xml` file from a previous installation, when returning it, make sure it is configured to allow 2 start/stop threads as shown below (see `<Host name="localhost" appBase="webapps" startStopThreads="2"/>`):

```
...
    <Engine name="Catalina"
defaultHost="localhost">
        <Host name="localhost" appBase="webapps"
startStopThreads="2" />
    </Engine>
...
```

g. If you installed Artifactory as a service, you now need to run the service

- i. For a Linux service, browse to `ARTIFACTORY_HOME/bin` and execute the following command **as root**: `ARTIFACTORY_HOME/bin/installService.sh [USER [GROUP]]`
- ii. For Windows service, browse to `%ARTIFACTORY_HOME%\bin` and run `InstallService.bat`.

Debian Installation

▼ Upgrading a Debian installation...

- a. Log in as root (or use `sudo su -`).
- b. Execute the following command:

```
dpkg -i $jfrog-artifactory-<oss|pro>-5.y.z.deb
```

Managing Configuration Files

When upgrading a **Debian** installation the upgrade process overwrites the following set of configuration files:

- *system.properties*
- *config.xml*
- *default*
- *logback.xml*
- *mimetypes.xml*
- All files under *opt/jfrog/artifactory/misc*
- All files under *opt/jfrog/artifactory/webapps*

If any of these files were modified, a backed up file will be created automatically with a notification in the upgrade log. If you need to restore the configuration changes you can restore them from the backup created.

Running Artifactory as ROOT

If you have configured Artifactory to run as the ROOT application in Tomcat, before you proceed, you need to follow the steps described in [this Knowledge Base article](#).

Upgrading to version 5.4.0 and above

From version 5.4.0, Artifactory's management of Access Tokens was moved to a separate service, Access, installed as a separate web application under the same Tomcat as Artifactory. This requires your Tomcat's *server.xml* to be configured to allow running 2 processes. If you are using a *server.xml* file from a previous installation, when returning it, make sure it is configured to allow 2 start/stop threads as shown below (see `<Host name="localhost" appBase="webapps" startStopThreads="2"/>`):

```
...
    <Engine name="Catalina" defaultHost="localhost">
        <Host name="localhost" appBase="webapps"
startStopThreads="2" />
    </Engine>
...
```

Docker Installation

▼ [Upgrading a Docker installation...](#)

In order to keep your data and configuration between versions, when upgrading the Artifactory Docker image, you need to use an external mounted volume as described under [Managing Data Persistence](#).

To upgrade the Artifactory Docker image, follow these steps:

- a. Stop current container
- b. Start a container with new version, using same data and configuration
- c. Remove old container

The example below shows this process for upgrading Artifactory from v5.0.0 to v5.1.0.

```
$ # Stop the currently running container
$ docker stop artifactory-5.0.0

$ # Start a new container from the new version image
$ docker run -d --name artifactory-5.1.0
--volumes-from=artifactory-5.0.0 -p 8081:8081
docker.bintray.io/jfrog/artifactory-pro:5.1.0

$ # Remove old container
$ docker rm artifactory-5.0.0
```

Once these commands have completed successfully, you would have the new version (5.1.0 in the above example) running with the data and configuration from the old version that was removed.

Running Artifactory as ROOT

If you have configured Artifactory to run as the ROOT application in Tomcat, you need to follow the steps described in [this Knowledge Base article](#).

Upgrading to version 5.4.0 and above

From version 5.4.0, Artifactory's management of Access Tokens was moved to a separate service, Access, installed as a separate web application under the same Tomcat as Artifactory. This requires your Tomcat's server.xml to be configured to allow running 2 processes. If you are using a server.xml file from a previous installation, when returning it, make sure it is configured to allow 2 start/stop threads as shown below (see <Host name="localhost" appBase="webapps" **startStopThreads="2"/>**):

```
...
    <Engine name="Catalina" defaultHost="localhost">
        <Host name="localhost" appBase="webapps"
startStopThreads="2" />
    </Engine>
...
```

RPM Installation

▼ Upgrading an RPM installation...

Make sure you are upgrading from v3.6 or above

When running as an RPM installation, you can only upgrade to v5.x if your current version is 3.6 or above. If necessary, first upgrade your current version to 3.6, and then upgrade to v5.x .

If you try to upgrade a version below 3.6 using `rpm --force` you may end up deleting all of your data.

- Download the **Artifactory Pro RPM Installer**. The latest version can be downloaded from the [JFrog Artifactory Pro Download Page](#). Previous versions can be downloaded from [JFrog Bintray](#).
- Log in as root (or use `sudo su -`).
- Execute the following command:

```
rpm -U jfrog-artifactory-pro-5.y.z.rpm
```

Switching from Artifactory OSS to Pro

If you are just switching from Artifactory OSS to Pro with the same version number, you need to append the command with `--force --nodeps` as follows:

```
rpm -U jfrog-artifactory-pro-5.y.z.rpm --force --nodeps
```

During an upgrade of an RPM installation different files may get backed up, where the backup file is appended with either a `.rpmorig` or a `.rpmnew` extension.

A `.rpmorig` extension means that the original file in your installation, the one that was there before performing the upgrade, was backed up before being replaced in the upgrade process.

A `.rpmnew` extension means that the original file in your installation, was **not** replaced in the upgrade, and instead, the new file with the same filename was backed up.

In either case, Artifactory will display a message such as:

```
warning: /etc/opt/jfrog/artifactory/default saved as /etc/opt/jfrog/artifactory/default.rpmorig
```

In these cases we recommend comparing the file installed once the upgrade has been completed with the backed-up file to see which best fits your needs, and using that one in the final setup.

If you make any changes, you may need to restart Artifactory for the change to be applied.

Upgrading Using YUM

An easy way to upgrade Artifactory from version 3.x or 4.x to the latest version is to use YUM with the Bintray Artifactory repository. The code snippets below show how to do this depending on whether your current version is below 3.6, or 3.6 and above.

Upgrading an Artifactory HA cluster?

If you are upgrading an Artifactory HA cluster, and you are running with a version that is older than version **5.4.6**, you should review the instructions on [Upgrading an Enterprise HA Cluster](#) prior to upgrading.

▼ If your current version is 3.6 and above:

```
curl https://bintray.com/jfrog/artifactory-pro-rpms/rpm -o
bintray-jfrog-artifactory-pro-rpms.repo && sudo mv
bintray-jfrog-artifactory-pro-rpms.repo /etc/yum.repos.d
yum install jfrog-artifactory-pro
```

▼ If your current version is below 3.6:

```
curl https://bintray.com/jfrog/artifactory-pro-rpms/rpm -o
bintray-jfrog-artifactory-pro-rpms.repo && sudo mv
bintray-jfrog-artifactory-pro-rpms.repo /etc/yum.repos.d
yum upgrade artifactory
yum install jfrog-artifactory-pro
```

Running Artifactory as ROOT

If you have configured Artifactory to run as the ROOT application in Tomcat, before you proceed, you need to follow the steps described in [this Knowledge Base article](#).

Upgrading to version 5.4.0 and above

From version 5.4.0, Artifactory's management of Access Tokens was moved to a separate service, Access, installed as a separate web application under the same Tomcat as Artifactory. This requires your Tomcat's `server.xml` to be configured to allow running 2 processes. If you are using a `server.xml` file from a previous installation, when returning it, make sure it is configured to allow 2 start/stop threads as shown below (see `<Host name="localhost" appBase="webapps" startStopThreads="2"/>`):

```

...
    <Engine name="Catalina" defaultHost="localhost">
        <Host name="localhost" appBase="webapps"
startStopThreads="2" />
    </Engine>
...

```

4. Start up the secondary node.
5. Add the secondary node back to the load balancer.
6. Repeat this process for each secondary node.

Upgrading to version 5.2.1 and above and using an NFS mount?

Once you have completed an upgrade, if your new version is 5.2.1 and above, and you are using a shared NFS mount, make sure to remove the Bootstrap bundle archive from the `$NFS_MOUNT/ha-etc` folder.

Check your installation

After starting up each secondary node, we recommend inspecting the `ha-node.properties`, `db.properties` and `binary store.xml` files (under `$ARTIFACTORY_NODE_HOME/etc`) to ensure they are correctly configured

Verify the HA Installation and Configuration

Once you have completed upgrading your HA cluster, you can verify that your cluster has been installed and configured correctly use the following series of tests:

1. Directly Access the Artifactory UI for the server you have just configured
2. In the **Admin** module go to **Advanced | System Logs** to view the log and verify that you see an entry for **HA Node ID**.

```

Artifactory HA
Version: 5.x-SNAPSHOT
Revision: 526
Artifactory Home: '/home/daniela/artifactory-pro-4.12.2-B'
Artifactory data dir: 'null'
Artifactory backup dir: 'null'
HA Node ID: 'art-b'

```

3. The bottom of the module navigation bar should also indicate that you are running with an Enterprise licens. In case of an error you will see an error message in the page header.



4. Access Artifactory through your load balancer and log in as **Admin**.
5. In the **Admin** module go to **Configuration**. There should be a section called **High Availability**. When selected you should see a table with details on all the Artifactory nodes in your cluster as displayed below.

Configure High Availability

All Artifactory Nodes

ID	Start Time	URL	Membership Port	State	Role	Last Heartbeat	Version	Revision	Release Date
ha_artifactory_1_1	22-07-15 10:12:39 UTC	http://172.17.0.19:8081/artifactory	10042	Running	Primary	22-07-15 10:13:19 UTC	4.0.0	1677	22-07-15 09:45:32 UTC
ha_artifactory_1_2	22-07-15 10:13:17 UTC	http://172.17.0.20:8081/artifactory	10042	Running	Member	22-07-15 10:13:21 UTC	4.0.0	1677	22-07-15 09:45:32 UTC
ha_artifactory_1_3	22-07-15 10:13:17 UTC	http://172.17.0.21:8081/artifactory	10042	Running	Member	22-07-15 10:13:20 UTC	4.0.0	1677	22-07-15 09:45:32 UTC

< page 1 of 1 >

- In the **Admin** module under **Configuration | General**, verify that the **Custom URL Base** field is correctly configured to the URL of the Load Balancer.

Want to stop using NFS?

If you want to stop using a shared NFS once the upgrade procedure is complete (**this is optional**), please refer to [Migrating Data from NFS](#) to migrate to alternative storage.

Troubleshooting the Upgrade Process

If you run into any problems during the upgrade process, please refer to [Troubleshooting HA](#) for a possible resolution.

Using Artifactory

Overview

Artifactory provides you with the features you need to manage your binary repositories, both through an intuitive UI and with an extensive set of APIs.

The Artifactory UI is divided into four main modules:

Home

The **Home** module serves as a dashboard where you can easily access general information and commonly used features. For more details, please refer to [General Information](#).

Artifacts

The **Artifacts** module is used to browse the repositories in your system and search for artifacts using a number of different methods. While browsing through repositories, Artifactory displays useful information and provides you with tools to perform essential actions to manage your repositories.

Search

The **Search** module is where you can search for Artifacts by name, package type, properties and a variety of additional methods offered by Artifactory. For more details, please refer to [Searching for Artifacts](#).

Build

The **Build** module displays the **Build Browser** where you can view all the CI server projects that output their builds to Artifactory. Here you can drill down to view detailed build information and compare one build to another.

Admin

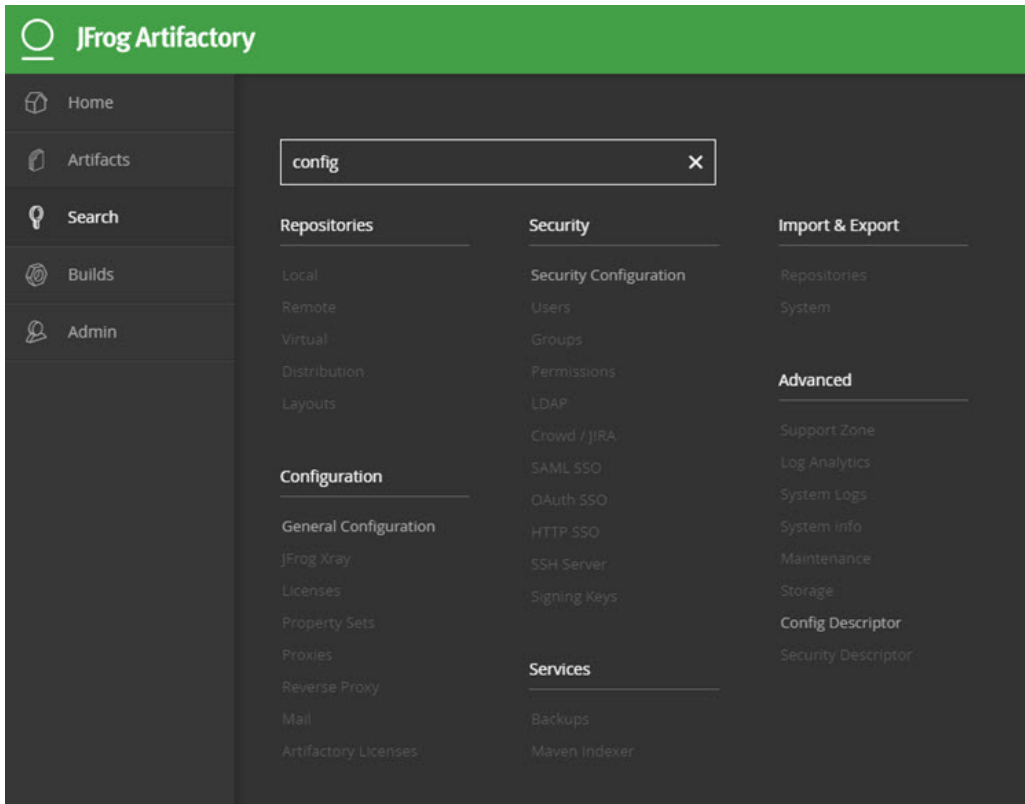
The **Admin** tab is only available to users who are defined as administrators in the system, and is used to

perform a variety of administration and maintenance activities such as:

- Configuring different entities such as repositories, software licenses, proxies, mail servers and more
- Managing different aspects of system security such as user definitions, groups, permissions, LDAP integration and more
- Managing backup and indexing of repositories
- Managing import and export of repositories or of the entire system
- Accessing system information and scheduling different cleanup operations

This functionality is available through the **Admin** menu which is displayed when a logged-in administrator hovers the mouse over **Admin** in the navigation bar.

To help locate a specific action, you can enter a search term in the **Filter** field and have Artifactory emphasize matching menu items.



In addition to the feature set offered by the UI, Artifactory provides an extensive [REST API](#) to facilitate integration with build automation and continuous integration systems.

Page Contents

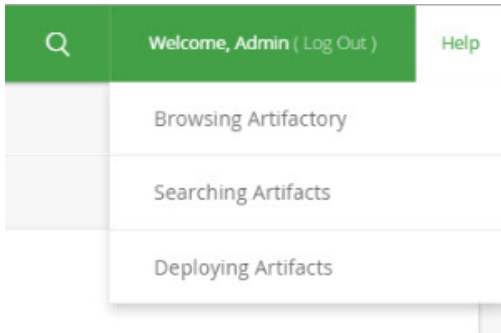
- [Overview](#)
- [Help Menu](#)
- [Set Me Up](#)
 - [Inserting User Credentials](#)
- [Keyboard Shortcuts](#)

Read more

- [Getting Started](#)
- [General Information](#)
- [Browsing Artifactory](#)
- [Using WebDAV](#)
- [Searching for Artifacts](#)
- [Deploying Artifacts](#)
- [Manipulating Artifacts](#)
- [Updating Your Profile](#)
- [Authentication](#)
- [Artifactory REST API](#)

Help Menu

On every screen of Artifactory, you can click the **Help** menu to display a list of help topics relevant for the current screen. Selecting any of the help topics opens a new browser tab displaying the corresponding Artifactory documentation.



Set Me Up

Artifactory's **Set Me Up** feature provides you with the commands you can use to deploy and resolve artifacts to and from specific repositories. Simply select any item in the [Tree Browser](#) or [Simple Browser](#) and click **Set Me Up**.

A screenshot of the "Set Me Up" dialog box in Artifactory. The dialog has a title bar with "Set Me Up" and a close button. Below the title bar, there are two dropdown menus: "Tool" with "Vagrant" selected and "Repository" with "vagrant" selected. To the right of these dropdowns is a green button labeled "Insert Credentials". Below the dropdowns, there are two sections: "Deploy" and "Resolve". The "Deploy" section contains the text "To deploy Vagrant boxes to this Artifactory repository using an explicit URL with Matrix Parameters use:" followed by a code block containing a curl command. The "Resolve" section contains the text "To provision a Vagrant box, all you need is to construct it's name in the following manner." followed by a code block containing a vagrant command. Both code blocks have a copy icon in the top right corner.

Regardless of what was selected when you pop up this dialog, you can select a **Tool** and a specific **Repository** (the list of repositories displayed corresponds to the tool you selected), and Artifactory will provide the relevant commands according to your selection.

Inserting User Credentials

Every **Set Me Up** dialog includes an "Insert Credentials" button. When pressed, Artifactory will prompt you for your password and will then replace generic credential placeholders used in code snippets with your own corresponding credentials.

Getting Started

Overview

To get you up and running as quickly and easily as possible for a new installation, Artifactory offers two ways to configure your basic initial setup:

- **Onboarding Wizard:** The onboarding wizard is invoked first time you start up Artifactory. It will take you through the steps of initial configuration using a convenient and intuitive UI.
- **YAML Configuration File:** The YAML configuration file offers an alternative way to specify your initial settings allowing you to skip the onboarding wizard.

In addition, you may use a combination of these two methods, specifying some of your initial setup in the YAML file, and then skipping the corresponding sections in the onboarding wizard.

The initial setup lets you configure the following basic settings:

- License
- Base URL (through the YAML configuration file only)
- Admin password (through the onboarding wizard only)
- Proxy server
- Initial default repositories

Page contents





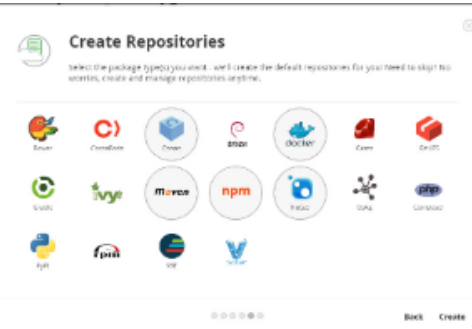
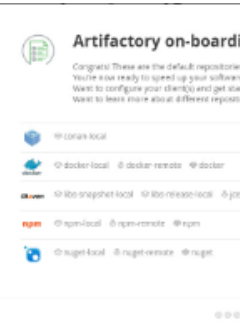
- Overview
- Onboarding Wizard
- YAML Configuration File
 - Limitations
 - Location and Usage
 - Exporting a Configuration

Bootstrapping only

Remember that both the onboarding wizard and the YAML configuration file can only be used to configure Artifactory upon bootstrapping it for the first time. Once Artifactory has been started up or a repository has been set up or used, the onboarding wizard is no longer accessible, and the YAML configuration file is not read.

Onboarding Wizard

The onboarding wizard makes sure you get Artifactory set up with the minimal information needed to get started.

		
<p>Welcome: The beginning of the onboarding wizard. Click Next to get started.</p>	<p>License: Enter your license key and click Next to continue.</p>	<p>Admin password: Set the admin password (recommended) and click Next to stay with the default admin password.</p>
		

Proxy: Configure a proxy server and click **Next** to continue, or click **Skip to configure one later**.

Create Repositories: Select the package formats for which Artifactory should create default repositories and click **Create** to continue.

Summary: Displays the default configuration according to your selection. Click **Next** to continue the wizard and get started with Artifactory.

YAML Configuration File

Setting up Artifactory using the YAML configuration file is a convenient alternative to going through the startup wizard. In addition, it gives you an easy way to save the basic configuration of one instance and then quickly and easily reproduce that configuration in other instances you set up.

When using the YAML configuration file, you don't have to configure all the parameters described in the [Overview](#) above. You may configure only some of the parameters using the YAML file, and then configure the others through the start up wizard, or manually later on after Artifactory has started up.

Limitations

These limitations stem from the principle that the YAML configuration file is designated for configuration of new Artifactory instances that essentially, have not been used before. When bootstrapping a new instance of Artifactory, it will load the configuration specified in this file if all of the following conditions are met:

- No repositories have been created
- A proxy has not been set up, or a proxy has been set up and you did not configure proxy setup through the YAML configuration file
- The base URL has not been set up, or the base URL has been set up and you did not configure the base URL through the YAML configuration file
- Artifactory has not been activated with a license, or it has been activated with a license and you did not configure the license through the YAML configuration file

Location and Usage

The YAML configuration file template can be found under `$ARTIFACTORY_HOME/misc/artifactory.config.template.yml`. To specify your initial bootstrap configuration, uncomment the relevant sections in the file and provide the configuration details. Rename the file, save it as `artifactory.config.import.yml` and place it under Artifactory's `etc` folder. When done you should have the following configuration file: `$ARTIFACTORY_HOME/etc/artifactory.config.import.yml`

An example of the YAML configuration file template used for Artifactory 5.0 can be found below:

▼ [YAML configuration file template...](#)

```
---
version: 1
## This file is complementary to the JFrog Artifactory startup wizard,
## and may be used to specify the initial basic
## settings for a new Artifactory installation, namely:
## * License Key(s)
## * Base URL
## * Proxy
## * Default repositories
##
##
## HOW TO USE THIS FILE:
##
## To import these settings when bootstrapping Artifactory, save this
## file as artifactory.config.import.yml under Artifactory's /etc folder
## Artifactory will load this file if all of the following conditions
## are met:
## - no repositories have been created
## - a proxy has not been set up, or you did set up a proxy
```

```
externally, but did not configure proxy setup through this file
## - the base URL has not been set up, or you did set up the base URL
externally, but did not configure the base URL setup through this file
## - Artifactory has not been activated with a license, or Artifactory
has been activated with a license, and you did not specify a license
in this file
##
## To have any of these parameters automatically configured when you
bootstrap an Artifactory instance using this file,
## simply uncomment the relevant sections below, and where required,
provide values.

#####
#####
# General Configurations #
#####
#####
GeneralConfiguration:
## License key to import in onboarding
  licenseKey : "Enter your license key"

## Setup the Artifactory base URL
## For more information about the Artifactory base URL, please refer
to
##
https://www.jfrog.com/confluence/display/RTF/Configuring+Artifactory#
ConfiguringArtifactory-GeneralSettings
## Uncomment the line below to set the Artifactory base URL
# baseUrl : "https://mycomp.arti.co"

## Configure proxies for artifactory
## For more information on configuring a proxy in Artifactory, please
refer to
## https://www.jfrog.com/confluence/display/RTF/Managing+Proxies
## Uncomment the lines below to setup a proxy
# proxies :
#   - key : "proxy1"
#   host : "https://proxy.mycomp.io"
#   port : 443
#   userName : "admin"
#   password : "password"
#   defaultProxy : true
#   - key : "proxy2"
#   ...
#####
#####
# Onboarding Configurations #
#####
#####
OnboardingConfiguration:
## Uncomment the package types for which you want to create default
repositories
  repoTypes :
```

```
# - bower
# - cocoapods
# - conan
# - debian
# - docker
# - gems
# - gradle
# - ivy
# - maven
# - npm
# - nuget
# - opkg
# - composer
# - pypi
# - sbt
```

```
# - vagrant
# - rpm
# - gitlfs
```

For example, to set your base URL to be "https://acme.artifactory.com", you should uncomment the **baseUrl** section and, **while keeping the same indentation**, set:

```
baseUrl : "https://acme.artifactory.com"
```

Indentation

Indentation is significant in YAML. Make sure to maintain the same indentation levels when editing the YAML configuration file.

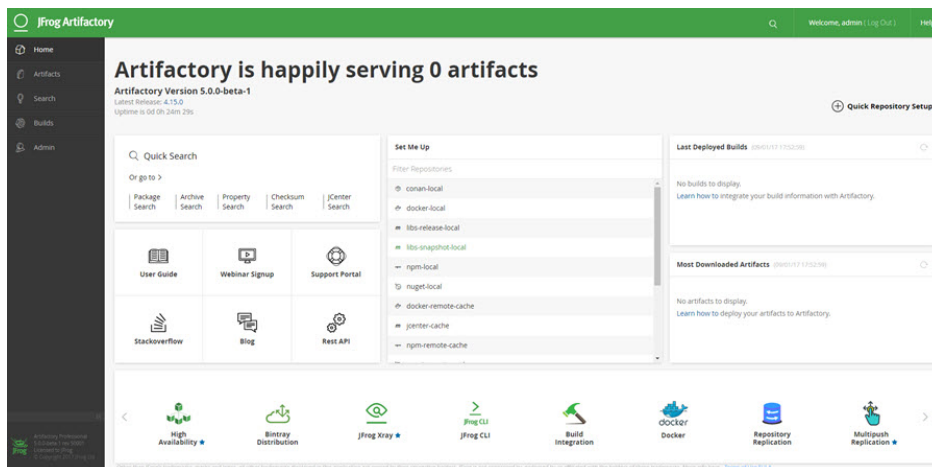
Exporting a Configuration

When Artifactory is bootstrapped for the first time, it stores a copy of its initial configuration under `$ARTIFACTORY_HOME/etc/artifactory.config.<timestamp>.yaml` regardless of whether it was bootstrapped using the [Onboarding Wizard](#), or using a YAML configuration file. To use this configuration to bootstrap additional Artifactory instances, copy the file into the new instance's `$ARTIFACTORY_HOME/etc` folder and rename it to `artifactory.config.import.yaml`.

General Information

Overview

The **Home** screen is divided into convenient sections and panels that provide useful information or quick links to commonly used features.

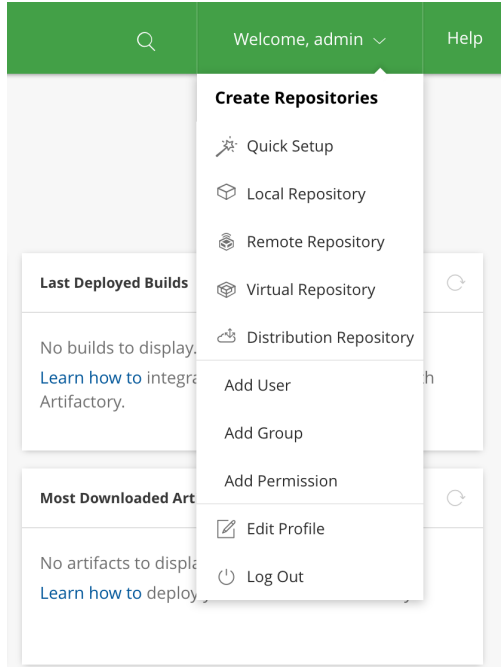


Page contents

- Overview
 - Top Ribbon
 - Basic Information
 - Search Panel
 - Set Me Up
 - Download and Deploy Statistics
 - Quick Repository Setup
 - Reference Panel
 - Features Panel

Top Ribbon

The top ribbon in Artifactory displays the logged-in user, and provides access to Quick Search and Help. Admin users may click the user name to display a menu of common actions for easy access.



Basic Information

At the top of the Home Screen, Artifactory displays basic information such as total number of artifacts, version information and uptime.

Search Panel

The Search Panel provides quick links to the different searches you can perform in Artifactory. For more details, please refer to [Searching for Artifacts](#).

Set Me Up

The Set Me Up panel provides quick access to details on how to configure your different clients to work with the corresponding repositories you have created. Just click one of the repositories to view its "Set Me Up" screen.

Set Me Up

Tool
⇌ Docker

Repository
docker-local

🔒 Type password to insert your credentials to the code snippets

 →

General
Using Docker with Artifactory requires a reverse proxy such as Nginx or Apache. For more details please visit our [Docker Repositories](#) documentation.

In this example we use **artprod.company.com** to represent the Docker repository in Artifactory.
To login use the *docker login* command.

```
1 docker login artprod.company.com
```

And provide your Artifactory username and password or API key.
If anonymous access is enabled you do not need to login.
To manually set your credentials, or if you are using Docker v1, copy the following snippet to your `~/.docker/config.json` file.

```
1 {
2   "auths": {
3     "https://artprod.company.com": {
4       "auth": "<USERNAME>:<PASSWORD> (converted to base 64)",
5       "email": "youremail@email.com"
6     }
7   }
}
```

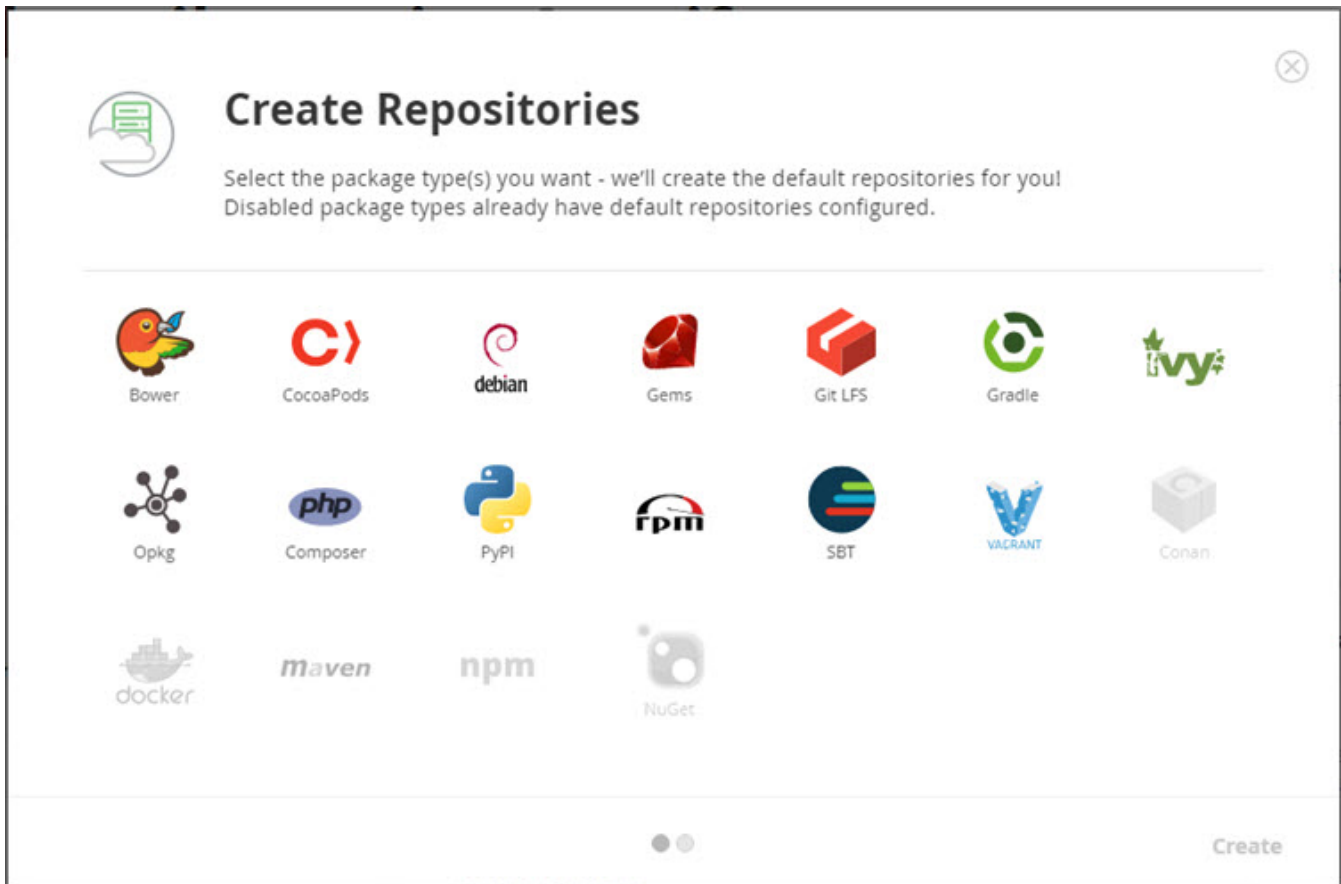
For more details, please refer to [Set Me Up](#).

Download and Deploy Statistics

These panels provide information on your last deployed builds and most downloaded artifacts.

Quick Repository Setup

The "Quick Repository Setup" button offers a quick and easy way to set up default repositories for any of the supported package formats for which you have not yet set up repositories.



Reference Panel

The Reference Panel provides links to useful references such as the [JFrog Artifactory User Guide](#) (this guide) and [Artifactory's REST API documentation](#), as well as additional useful links such as the [Support Portal](#), [blog](#), [webinar signup](#) page and questions tagged with "Artifactory" on [Stack Overflow](#).

Features Panel

The Features Panel at the bottom of the screen displays all the features available to you according to the license you have used to activate Artifactory. Hover over any feature to get quick links to more information and a link to the relevant section in this user guide

Browsing Artifactory

Overview

The **Artifacts** module in Artifactory displays the **Artifact Repository Browser** that provides two ways to browse through repositories:

- **Tree Browser**: Displays the repository as a tree
- **Simple Browser**: Focuses on the currently selected item and displays the level below it in the repository hierarchy as a flat list

Both browsers adhere to the security rules defined in the system, and only allow you to perform actions authorized by the system security policies.

To switch between browsing modes, simply select the corresponding link at the top of the **Artifact Repository Browser**.

Artifact Repository Browser

Tree Simple 

Compress Empty Folders

 alonn-local

▶  bower-local

Both the Tree Browser and Simple Browser have features to help you navigate them and search for repositories:

- **Repository type icon:** Each repository is displayed with a distinct icon that represents its type (local, cache, remote and virtual).
- **Search in the browser:** You can search for a specific repository in both browsers by clicking on the filter icon.
- **Keyboard navigation:** While in the browser, type the name of the repository you are searching for and Artifactory will navigate you to that repository.
- **Filters:** Click the filter icon to filter the repositories displayed in the browser to only display the types that interest you. You can also simply type the filter expression while on the browser.

Page Contents

- Overview
- Tree Browsing
 - [Sorting the Tree Browser](#)
- Simple Browsing
- Filters
- Information Tabs
- List Browsing
- Remote Browsing
- Trash Can
- WebDAV Browsing

Tree Browsing

The Tree Browser lets you drill down through the repository hierarchy and displays full information for each level within it. For any repository, folder or artifact selected in the tree, a tabbed panel displays detailed data views and a variety of actions that can be performed on the selected item. The information tabs available are context sensitive and depend on the item selected. For example, if you select an npm package, an Npm Info tab displays information specific to Npm packages. Similarly for NuGet, RubyGems and any other supported package formats.

Collapse All

Click on the "Tree" link at the top of the tree browser to collapse all open nodes in the tree.

Sorting the Tree Browser

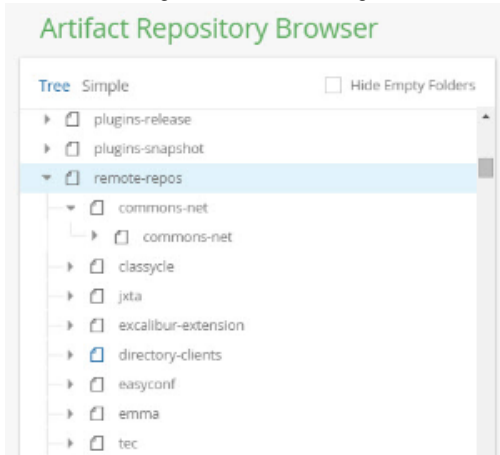
The default order in which repositories appear in the Tree Browser is: Distribution, Local, Remote, Virtual.

You can modify this order through the `artifactory.treebrowser.sortRepositories.sortByType` system property.

For example, to reverse the order, you would set:

```
artifactory.treebrowser.sortRepositories.sortByType=virtual,remote,local,distribution
```

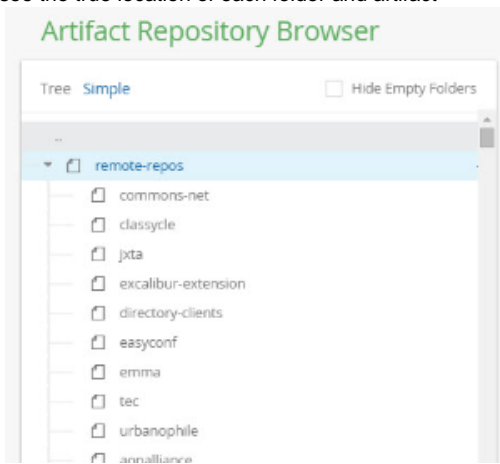
If you omit any repository type in the specified sort order, it will be ordered according to the default setting.



Simple Browsing

The simple browser lets you browse repositories using simple path-driven URLs, which are the same URLs used by build tools such as Maven. It provides lightweight, view-only browsing.

A unique feature of this browsing mode is that you can also [view remote repositories](#) (when enabled for the repository), and virtual repositories to see the true location of each folder and artifact



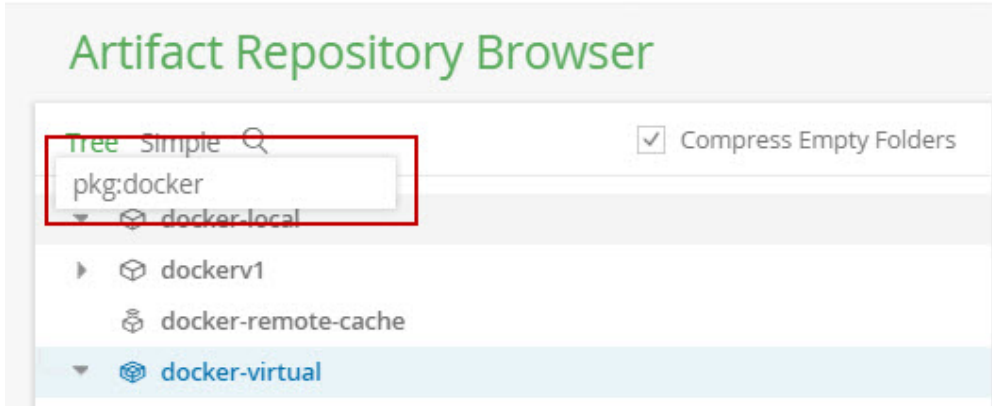
Filters

You can apply a filter to both the Tree Browser and the Simple Browser, either by clicking the search icon, or just typing out the filter expression while on the browser.

By repository name: To filter by repository name, just type the name you want to filter by

By package type: To only display repositories for a particular package type, type **pkg:<package type>**. For example, typing **pkg:docker** will filter out any repositories that are not Docker repositories

By repository type: To only display particular repository types, type **repo:<repository type>**. For example, typing **repo:local** will filter out any repositories that are not local repositories. The options are **local**, **cached**, **remote** and **virtual**.



Information Tabs

Selecting any item in the Tree or Simple browser displays tabs which provide information regarding the metadata associated with the selected item:

General	<p>Info: General Information including download statistics such as the total number of downloads, time stamp of last download and the last user who downloaded.</p> <p>Xray: Indicates if, and when the last time the selected artifact was indexed by JFrog Xray, as well as the Top Severity reported for the selected item and when the reporting alert was last updated.</p> <p>Dependency Declaration: For Maven artifacts, this section provides code snippets for common build tools' dependency declaration</p> <p>Virtual Repository Associations: Indicates which virtual repositories "contain" the selected artifact.</p> <p>Checksums displays SHA1, SHA-256 and MD5 checksums automatically.</p>
Effective Permissions	<p>The permissions (Delete/Overwrite, Deploy/Cache, Annotate, Read) that each user or group regarding the selected item. Permissions for groups are as specifically assigned to them. Permissions for individual users are the union of permissions specifically assigned as well as those inherited by virtue of the user being included in groups.</p>
Properties	<p>The list of properties annotating the selected item.</p>
Watchers	<p>The list of users watching this item.</p>
Builds	<p>The list of builds that either produce or use the selected item.</p>
Governance	<p>Information on the selected item retrieved from Black Duck Code Center.</p>
Xml/Pom View	<p>XML and POM files also display a tab through which you can view the file contents.</p>

artifactory-pro-war-4.x-20160616.132515-1.war

Download Actions

General Effective Permissions Properties Watchers Builds Governance

Created: 16-06-16 13:25:16 +00:00 ⓘ

Last Modified: 16-06-16 13:24:57 +00:00 ⓘ

Licenses: Not Found Add | Scan | Delete | Search Archive License File ⓘ

Downloaded: 6

Last Downloaded By: xray

Last Downloaded: 27-07-16 12:55:55 +00:00

Remote Downloaded: 0

Filtered ⓘ

Xray

Status: Indexed

Status Last Updated: 27-07-16 12:56:23 +00:00

Top Severity: **Minor**

Alert Last Updated: 27-07-16 12:56:25 +00:00

Dependency Declaration

Build Tool: **Maven** Ivy Gradle Sbt

```

1 <dependency>
2   <groupId>org.artifactory.pro/<groupId>
3   <artifactId>artifactory-pro-war/<artifactId>
4   <version>4.x-20160616.132515-1/<version>
5   <type>war/<type>
6 </dependency>

```

Virtual Repository Associations

libs-snapshot

Checksums

SHA-256: 99a6e6adbeb7781aaebc996e4c7cdf5cca5bc071ffd9f11793a72fa9fabd48e2

SHA-1: f8936f46d81280d2f10826bd3405243e520f1d1a (Uploaded: Identical)

MD5: 01f5f2dcf928d345d490c8d46cf25930 (Uploaded: Identical)

List Browsing

List Browsing lets you browse the contents of a repository outside of the Artifactory UI. It provides a highly responsive, read-only view and is similar to a directory listing provided by HTTP servers.

To use List Browsing, click the icon to the right of a repository name in the Simple Browser (indicated in red in the screenshot above).

Creating public views with List Browsing

List browsing can be used to provide public views of a repository by mounting it on a well-known path prefix such as `list` (see example below).

This allows the system administrator to create a virtual host that only exposes Artifactory's List Browsing feature to public users (while maintaining write and other advanced privileges), but limiting access to the intensive UI and REST API features for internal use only.

```
http://host:port/artifactory/list/repo-path
```

Index of download.eclipse.org/

Name	Last modified	Size
.index/	14-Jul-2015 15:19	-
forums/->	-	-
forums/->	-	-
forums/->	-	-
forums/->	-	-
projects/->	-	-
releases/	30-Apr-2015 11:31	-
technology/	30-Apr-2015 11:31	-
donate->	-	-
downloads->	-	-
IRC->	-	-
IRC->	-	-

Artifactory/4.x-SNAPSHOT Server at 10.100.1.110 Port 8081

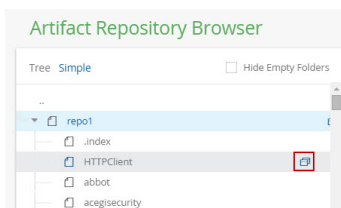
Remote Browsing

Artifactory remote browsing capabilities let you navigate the contents of the remote repository even if the artifacts have not been cached locally.

To enable remote browsing, you need to set the **List Remote Folder Items** checkbox in the remote repository configuration. Once this is set you can navigate that repository using the Simple or List Browser.

In the Simple Browser, an item that is not cached is indicated by an icon on its right when you hover over the item. In the List Browser, an item that is not cached is indicated by an arrow.

Simple Browser



List Browser

Index of repo1/acegisecurity

Name	Last modified	Size
../		
acegi-security/	30-Apr-2015 11:31	-
acegi-security-adapters/->	-	-
acegi-security-cas/->	-	-
acegi-security-catalina/->	-	-
acegi-security-catalina-common/->	-	-
acegi-security-catalina-server/->	-	-
acegi-security-domain/->	-	-

Initial responsiveness of remote repositories

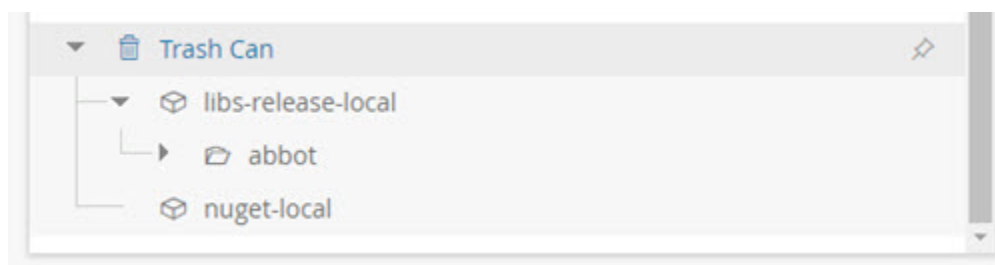
Initial remote browsing may be slow, especially when browsing a virtual repository containing multiple remote repositories. However, browsing speeds up since remote content is cached for browsing according to the [Retrieval Cache Period](#) defined in the remote repository configuration panel.

Trash Can

Artifactory provides a trash can as an easy way to recover items that have been inadvertently deleted from local repositories.

The trash can can be enabled or disabled in the [Trash Can Settings](#) by an Artifactory administrator.

When enabled, the trash can is displayed at the bottom of the Artifact Repository Browser and it holds all artifacts or repository paths that have been deleted from local repositories for the period of time specified in the **Retention Period** field.



Right-click on an item in the trash can gives you the option to **Refresh**, **Restore** it to its original location, or **Delete permanently**.

Right-click on the trash can icon gives you the option to **Refresh** the whole trash can, **Search Trash** for specific items, or **Empty Trash** which means that all items in the trash can will be permanently deleted.

Click the pin icon to pin the trash can so it remains visible even if you scroll the tree.

WebDAV Browsing

Artifactory fully supports browsing with WebDAV. For full details please refer to [Using WebDAV](#).

Using WebDAV

Overview

Artifactory supports WebDAV shares. A local or cached repository may be mounted as a secure WebDAV share, and made accessible from any WebDAV-supporting file manager, by referencing the URL of the target repository as follows:

```
http://host:port/artifactory/repo-path
```

When trying to deploy a file through WebDAV where file locking is enabled, the Artifactory log may display the following message:

```
"Received unsupported request method: lock".
```

In some cases this can be solved by disabling file locking before mounting the repository and is done differently for each WebDAV client. For example, for davfs2 file locking is disabled as follows:

```
echo "use_locks 0" >> /etc/davfs2/davfs2.conf
```

Note that while for some clients file locking is disabled by default, it is not necessarily possible to disable file locking in all clients.

Page Contents

- [Overview](#)
- [Authentication for davfs2 Clients](#)

- Authentication for Windows and other WebDAV clients

Authentication for davfs2 Clients

Davfs2 does not use preemptive authentication. Therefore, in order to authenticate using the user credentials, the client must be authenticated using two requests. The first request is sent without credentials, and receives a 401 challenge in response. Then, a second request is sent, this time with the credentials.

Anonymous access with Artifactory

Artifactory may be configured to allow anonymous access and it will therefore accept requests without authentication.

In this case, Artifactory will not respond with a 401 challenge and you will get file access with anonymous user permissions which may be less than your own user permissions.

To access your repository through Artifactory with your full user permissions you need add an authorization header to the client configuration. This way, the requests sent to Artifactory will be authenticated and there is no need to receive a 401 challenge and respond with a second request.

Thus you are given anonymous access to Artifactory, and yet can still authenticate with your own credentials.

This can be done as follows:

1. Encode your username and password credentials in base64 using the following Groovy script:

Groovy script

```
Basic ${"username:password".bytes.encodeBase64() }
```

2. Edit the file `/etc/davfs2/davfs2.conf` or `~/.davfs2/davfs2.conf` and add the encoded credentials to the authorization header as follows:

Adding authorization header

```
add_header Authorization "Basic c2hheTpwYXNzd29yZA=="
```

Authentication for Windows and other WebDAV clients

We suggest utilizing a tool such as [Cyberduck](#) (Open Source) when using Windows (see the note below) with WebDAV shared Artifactory repositories.

Limitation

Although the use of Windows WebDAV/WebClient components to map/mount a Windows Drive for a WebDAV shared Artifactory does provide a listing of the files - other operations such as copy/move operations are utilizing WebDAV commands which are not supported by Artifactory.

Searching for Artifacts

Overview

Artifactory provides several Search Types you can use to search for artifacts in the **Search** module:

- **Quick:** Search by artifact file name.
- **Package:** Search for artifacts according to the criteria specific to the package format.
- **Archive Entries:** Search for files that reside within archives (e.g. within a jar file).
- **Property:** Search for artifacts based on names and values of properties assigned to them.
- **Checksum:** Search for artifacts based on their checksum value.

- **JCenter**: Search for artifacts in Bintray's JCenter repository.
- **Trash Can**: Search for artifacts in Artifactory's trash can

Additional advanced search features are available through the [REST API](#).

Search in Artifactory provides true real-time results that always reflect the current state of the repository with no need to periodically rebuild indexes. Therefore, search results will immediately show any artifacts deployed, and will not show any artifacts removed. The * and ? wildcards are supported in your search term to help you narrow down your search. After conducting a search, you can hover over any result item for available actions such as:

Download	Download the artifact
Show in Tree	Displays the artifact within the Tree Browser where you can view its full details
Delete	Delete the artifact

Using wildcards

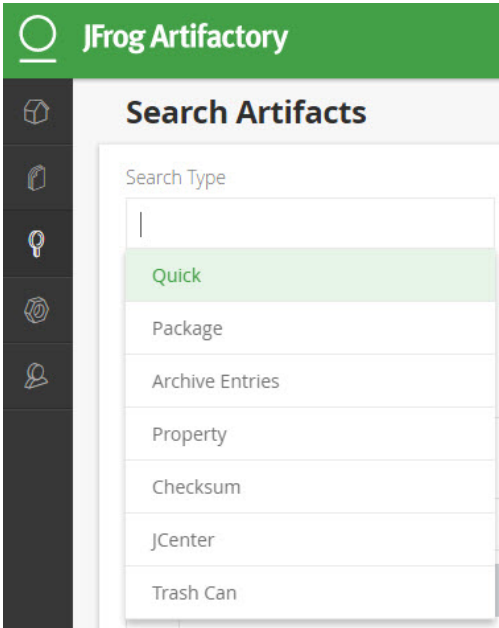
When searching with the Artifactory UI, Artifactory performs prefix matching for search terms in all the different search modes. For example, searching for `jfrog` is equivalent to searching for `jfrog*`. You can still use the * and ? wildcards by placing in your search term in double-quotes to help you narrow down your search (for example, `"a*.jar"`).

Page Contents

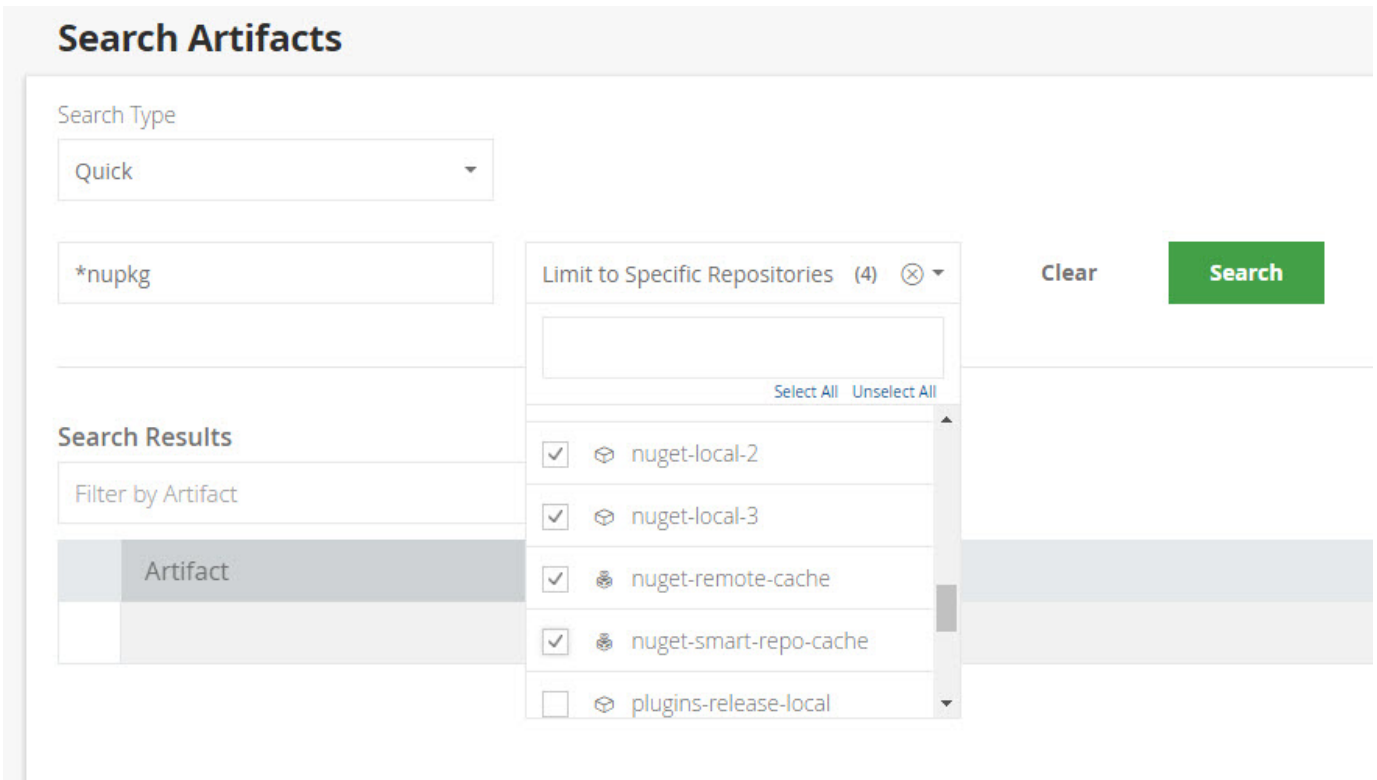
- [Overview](#)
- [General](#)
- [Quick Search](#)
- [Archive Search](#)
- [Package Search](#)
- [Property Search](#)
- [Checksum Search](#)
- [JCenter](#)
- [Trash Can](#)
- [Search Results Stash](#)

General

The different search features are available in the **Search** module. To start a search, simply select the search method you want to use.



Each search method offers a set of input fields corresponding to the type of search you have selected to help narrow down your search. For example, you can always narrow down your search by Selecting **Limit to Specific Repositories** as one of your search criteria.



Case Sensitive

For all searches, the search term is case-sensitive.

Quick Search

Using Quick Search you can search for artifacts by name. Select **Quick**, enter your search term and then click the "Search" button.

Search Artifacts

Search Type
Quick

*nupkg Limit to Specific Repositories (4) Clear Search

Search Results - 84 Items | [Stash Results](#) ?

Filter by Artifact

Artifact	Path	Repository	Modified
Angara.Serialization.0.2.0.0.nupkg	[root]	nuget-remote-cache	13-04-16 13:28:49 +03:00
Package250.1.0.0.nupkg	[root]	nuget-local-2	15-11-16 11:37:48 +02:00
Package250.1.0.1.nupkg	[root]	nuget-local-2	15-11-16 11:37:49 +02:00
Package250.1.0.10.nupkg	[root]	nuget-local-2	15-11-16 11:37:56 +02:00

For readability, you can limit the number of results displayed by setting the following two parameters in the `$ARTIFACTORY_HOME/etc/artifactory.system.properties` file:

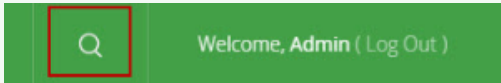
Configuring the number of search results

```
## Maximum number of results to return when searching through the UI
#artifactory.search.maxResults=500

## The backend limit of maximum results to return from sql queries issued
by users. Should be higher than maxResults.
#artifactory.search.userQueryLimit=1000
```

You can run a Quick Search from any screen

You can also run a Quick Search from any screen using the search field in the top-right corner of the screen.



Archive Search

Archive search performs a search for all files that match your search parameters in an archive. Typical examples are a zip or jar file, however, all file types defined in the [MIME types](#) configuration are supported. You can specify the following parameters for your search:

Name	The term to search for within the file name.
Path	Allows you to specify a path filter for the search.
Search class resources only	When checked, only class resources are searched. Any other file type is filtered out of the search.
Exclude Inner Classes	When checked, inner classes are excluded from the search.
Limit to Specific Repositories	Limits search to the specified repositories.

Search Artifacts

Search Type

Archive Entries

jar

Add search criteria...

Clear

Search

Path

Search Class Resources Only

Exclude Inner Classes Clear

Limit to Specific Repositories

Search Results

Filter by Name

Name

Artifact

View the source file

You can hover over a class file and select **View** to view the corresponding source file if it's available.

Package Search

Package search enables you to run a search based on a specific packaging type. For each type, you can specify search parameters based on the relevant metadata for the selected package type. For example, **Docker** search is suitable for searching through Docker repositories.

Search Artifacts

Search Type

Package

Select Package Type...

Bower

CocoaPods

Composer

Conan

Debian

Docker

Gems

Search Results

Filter by Name

Name

act

The following table displays the parameters you may use for each package type:

Search type	Search parameters
Bower	Package name, Version

CocoaPods	Package name, Version
Composer	Package name, Version
Conan	Package name, Version, User , Channel, OS, Architecture, Build Type, Compiler
Debian	File name (without the <i>.deb</i> extension), Distribution, Component, Architecture
Docker	Full Image Namespace, Image Tag, Image Digest
Gems	Package name, Version
Maven GAVC	Group ID, Artifact ID, Version, Classifier
Npm	Package name, Version, Scope
NuGet	Package ID, Version
Opkg	Package name, Version, Architecture, Priority, Maintainer
PyPI	Package name, Version
RPM	Package name, Version, Architecture, Release
Vagrant	Box Name, Version, Provider

All these search fields support the "?" and "*" wildcard characters.

Package search as an AQL query

For most package formats, package search is implemented as an AQL query. Click the "AQL" button to view the AQL query used in the search. You may also click the "Copy" icon in the AQL code snippet to copy the query to your clipboard.

Limit search to specific repositories

When limiting search to specific repositories, Artifactory will only let you select repositories with the corresponding package type. Package search depends on those repositories having the correct layout. Searching through repositories with the wrong layout will have unpredictable and unreliable results.

The example below shows the results of searching for any Docker image with "mysql" in its name:

Search Artifacts

Search Type: Package | Docker

mysql | Tag | Add search criteria... | Clear | Search

Search Results - 1 Items | Stash Results ?

Filter by Image | AQL | Delete | Page 1 of 1

Image	Tag	Repository	Modified
mysql	1	docker-local	Mon Aug 22 09:53:16 IDT 2016

Under the hood

Package search is based on standard properties that Artifactory attaches to packages according to their type. For example, when searching for NuGet packages, Artifactory is actually matching the search terms to the values for the `nuget.id` and `nuget.version` properties that should be attached to every NuGet package.

Limitation

Package search does not currently work on remote repository caches for RubyGems and Debian repositories.

Property Search

Artifactory Pro allows you to search for artifacts or folders based on [Properties](#) assigned to them, whether they are standard properties assigned by Artifactory, or custom properties which you can freely assign yourself.

To define your search parameters, in the **Key** field, enter the name of the property to search for, or select one from the list provided.

Then, in the **Value** field, set the value you are searching for in the specified property. To add more properties to the search use the **Add search criteria** drop list.

You can repeat this process to specify any number of properties and values for your search.

Wildcards can be used in the Property Value field

You can use the "?" or "*" wildcards in the **Value** field.

Combining properties and values

Properties are combined using the AND operator.

Different values assigned to a specific property are also combined using the AND operator.

This means that only artifacts that meet all the search criteria specified will be found.

The example below shows a search for artifacts that have a `build.number` property with a value of 2

Search Artifacts

Search Type: Property

build.number | 2 | Add search criteria... | Clear | Search

Search Results - 16 Items | Stash Results ?

Filter by Item | Delete | Page 1 of 1

Item	Type	Path	Repository	Modified
multi-2.17-20160629.123554-5.pom	□	maven/multi/2.17/org/jfrog/test/multi/2.17-SNAPSHOT	dist-repo	29-06-16 15:35:57 +03:00
multi-2.17-20160629.123554-5.pom	□	org/jfrog/test/multi/2.17-SNAPSHOT	libs-snapshot-local	29-06-16 15:35:57 +03:00
multi1-2.17-20160629.123554-5-sources.jar	□	maven/multi1/2.17/org/jfrog/test/multi1/2.17-SNAPSHOT	dist-repo	29-06-16 15:35:57 +03:00
multi1-2.17-20160629.123554-5-sources.jar	□	org/jfrog/test/multi1/2.17-SNAPSHOT	libs-snapshot-local	29-06-16 15:35:57 +03:00
multi1-2.17-20160629.123554-5.jar	□	maven/multi1/2.17/org/jfrog/test/multi1/2.17-SNAPSHOT	dist-repo	29-06-16 15:35:57 +03:00
multi1-2.17-20160629.123554-5.jar	□	org/jfrog/test/multi1/2.17-SNAPSHOT	libs-snapshot-local	29-06-16 15:35:57 +03:00

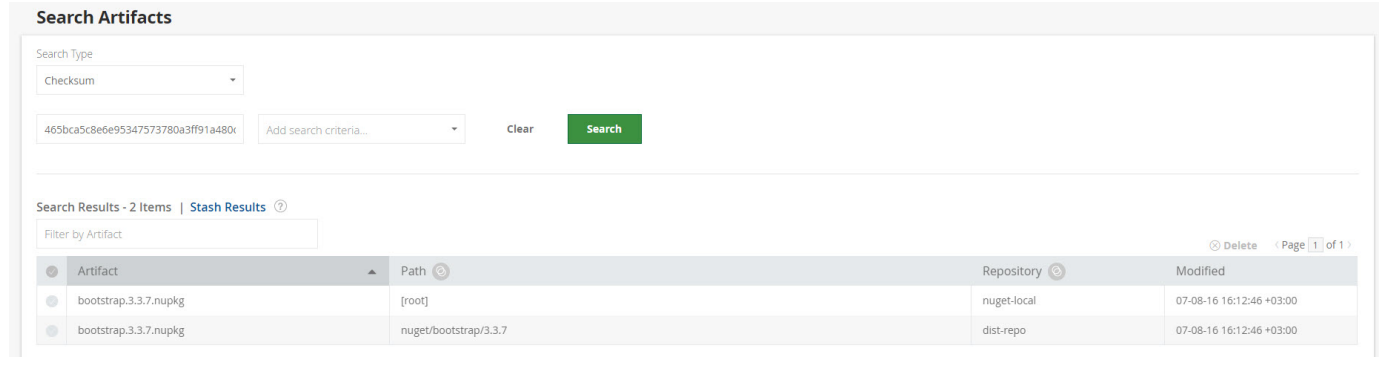
Checksum Search

Artifactory allows you to search for artifacts based on MD5, SHA1 or SHA2 checksum value.

This can be especially useful if you want to identify an artifact whose name has been changed.

Wildcard characters are not supported in Checksum Search, so the term entered in the search field must be valid MD5 or SHA1 value.

The example below shows a search for an artifact using its SHA1 checksum.



The screenshot shows the 'Search Artifacts' interface. The 'Search Type' is set to 'Checksum'. The search criteria field contains the SHA1 checksum '465bca5c8e6e95347573780a3ff91a480c'. The search results show 2 items:

Artifact	Path	Repository	Modified
bootstrap.3.3.7.nupkg	[root]	nuget-local	07-08-16 16:12:46 +03:00
bootstrap.3.3.7.nupkg	nuget/bootstrap/3.3.7	dist-repo	07-08-16 16:12:46 +03:00

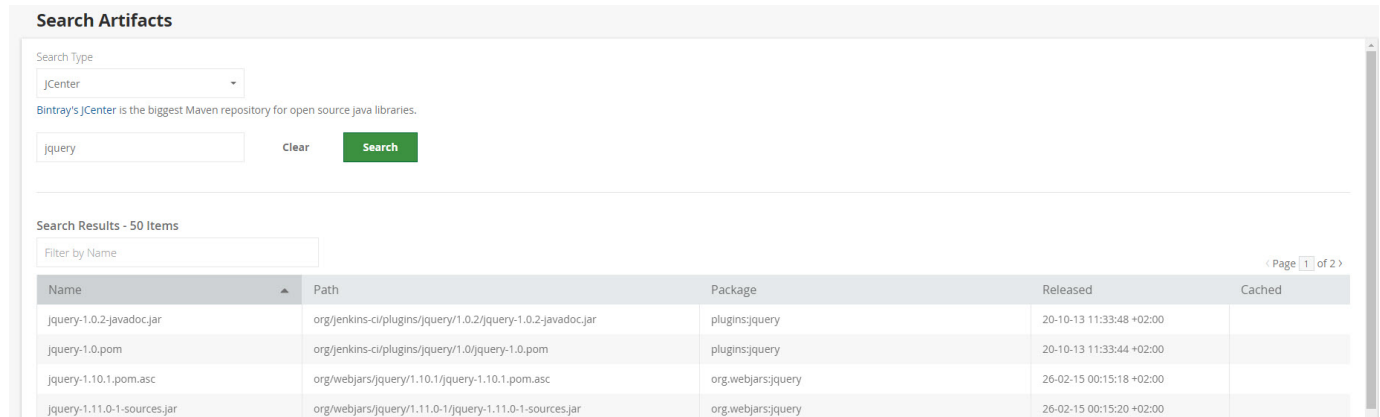
JCenter

Bintray is JFrog's software distribution platform. Using this free cloud-based service, you can publish, download and share your binaries with the developer community.

For more details, please refer to the [JFrog website Bintray page](#).

Artifactory provides a direct connection to Bintray's JCenter repository which contains a comprehensive collection of popular Apache Maven packages.

To search for packages on Bintray, select **JCenter** as the **Search Type** and enter the name of the package you are looking for.



The screenshot shows the 'Search Artifacts' interface with 'JCenter' selected as the search type. The search criteria field contains 'jquery'. The search results show 50 items:

Name	Path	Package	Released	Cached
jquery-1.0.2-javadoc.jar	org/jenkins-ci/plugins/jquery/1.0.2/jquery-1.0.2-javadoc.jar	plugins:jquery	20-10-13 11:33:48 +02:00	
jquery-1.0.pom	org/jenkins-ci/plugins/jquery/1.0/jquery-1.0.pom	plugins:jquery	20-10-13 11:33:44 +02:00	
jquery-1.10.1.pom.asc	org/webjars/jquery/1.10.1/jquery-1.10.1.pom.asc	org.webjars:jquery	26-02-15 00:15:18 +02:00	
jquery-1.11.0-1-sources.jar	org/webjars/jquery/1.11.0-1/jquery-1.11.0-1-sources.jar	org.webjars:jquery	26-02-15 00:15:20 +02:00	

You can hover over any search result and click **Show in Bintray** to display the selected artifact in Bintray.

Trash Can

Artifactory lets you search for artifacts you have "removed" to the trash can by selecting **Trash Can** as the **Search Type**.

Enter the artifact's name in the **Query** field, or use **Add search criteria...** and enter the artifact's checksum.

Search Artifacts

Search Type

Trash Can

465bca5c8e6e95347573780a3ff91a480c

Checksum Search

Clear

Search

Search Results Stash

Requires Artifactory Pro

This feature is available with an Artifactory Pro license

Artifactory maintains a stash where you can save search results. This provides easy access to artifacts found without having to run the search again and also provides a convenient way to perform bulk operations on the result set.

For details, please refer to [Saving Search Results in the Stash](#).

Deploying Artifacts

Overview

You can deploy artifacts into a local repository of Artifactory from the **Artifacts** module by clicking **Deploy** to display the **Deploy** dialog. Artifacts can be deployed *individually* or in multiples.

Search type: Quick Package Archive Property Checksum Remote

Artifact Repository Browser [Set Me Up](#) [Deploy](#)

Using Import to "deploy" a whole repository

If you want to "deploy" a whole repository, you should actually import it using the [Import Repository](#) feature in the **Admin** tab under **Import & Export | Repositories**.

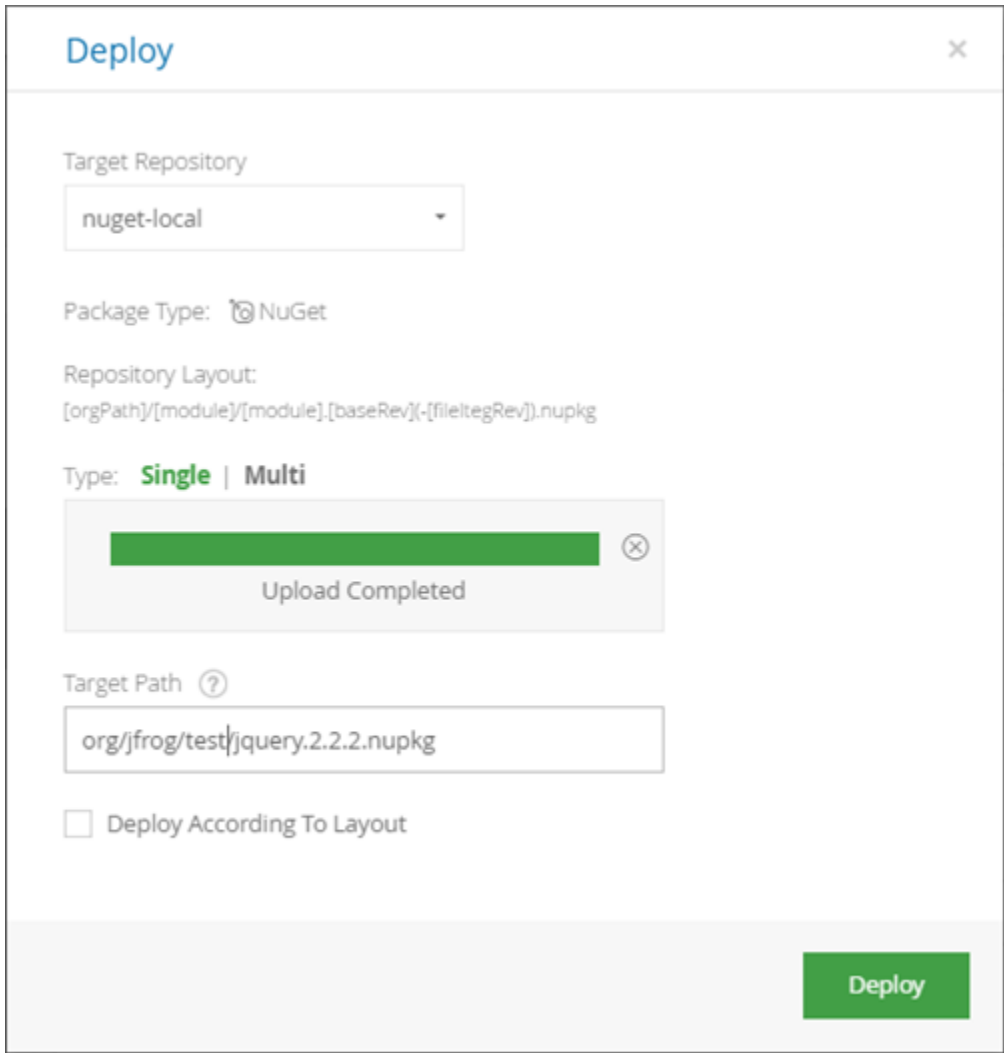
You can also deploy artifacts to any repository using Artifactory's REST API, see [this example](#) for a quick start.

Page Contents

- Overview
- Deploying a Single Artifact
 - Deploying According to Layout
 - Deploying Maven Artifacts
 - Deploying with Properties
 - Deploying with Multiple Properties
- Deploying an Artifact Bundle
- Deploying to a Virtual Repository
- Failed Uploads

Deploying a Single Artifact

To deploy a single artifact, simply fill in the fields in the Deploy dialog and click "Deploy".



Deploying According to Layout

The **Deploy** dialog displays the repository package type and layout configured. To deploy your package according to the configured layout, check **Deploy According to Layout**.

Artifactory displays entry fields corresponding to the layout tokens for you to fill in.

Deploy

Target Path ?

jfrog/test/test.2.2.2-jquery.nupkg

Deploy According To Layout

Layout Tokens

orgPath *

jfrog

module *

test

baseRev *

2.2.2

fileIntegRev

jquery

Deploy

If you are deploying a Maven artifact, you may need to configure additional attributes as described in the next section.

Suggested Target Path

Artifactory will suggest a **Target Path** based on the details of your artifact (this works for both Maven and Ivy). For example, if a JAR artifact has an embedded POM under its internal `META-INF` directory, this information is used.

Deploying Maven Artifacts

If you are deploying an artifact that conforms to the Maven repository layout, you should set **Deploy as Maven Artifact** to expose fields that specify the corresponding Maven attributes - **GroupID**, **ArtifactID**, **Version**, **Classifier** and **Type**.

The fields are automatically filled in according to the artifact name, however you can edit them and your changes will also be reflected in the **Target Path**.

If your target repository does not include a POM, set **Generate Default POM/Deploy Jar's Internal POM**, to use the POM within the artifact you are deploying, or generate a default POM respectively.

Take care when editing the POM manually

If you are editing the POM manually, be very careful to keep it in a valid state.

Deploy



Target Path [?](#)

org/slf4j/slf4j-api/1.7.20/slf4j-api-1.7.20.jar

Deploy as Maven Artifact [?](#)

Maven Artifact

Group ID *

org.slf4j

Artifact ID *

slf4j-api

Version *

1.7.20

Classifier

Type *

jar

Deploy

Deploy

Version *

1.7.20

Classifier

Type *

jar

Generate Default POM / Deploy Jar's Internal POM

```

1  <?xml version="1.0" encoding="UTF-8"?><project
   xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
2
3  <modelVersion>4.0.0</modelVersion>
4
5  <parent>

```

Deploy

Deploying with Properties

Properties can be attached to the uploaded file by specifying them on the **Target Path**.

First, unset the **Deploy as Maven Artifact** check box, if necessary.

Then, in the **TargetPath** field, add the properties delimited from the path and from each other by semicolons.

For example, to upload an artifact with the property **qa** set to "passed", and **build.number** set to "102", use the following **Target Path**:

```
dir1/file.zip;qa=passed;build.number=102
```

Deploying with Multiple Properties

To deploy multiple values to the same key add the same key again with the new value, e.g. `key1=value1;key1=value2` will deploy the file with property `key1` with value of `value1,value2`.

For example, to upload a file with property `passed` and values `qa`, stress use the following **Target Path**:

```
dir1/file.zip;passed=qa;passed=stress
```

Deploying Multiple Files

To deploy multiple files together, simple set the deploy **Type** to **Multi**, fill in the rest of the fields in the dialog and click "Deploy".

Deploy ×

Target Repository
libs-release-local

Repository Type: Maven

Type: Single **Multi**

Drop files or Select files

- artifactory-papi-2.6.4-javadoc.jar ⊗
- slf4j-api-1.7.5.jar ⊗

Target Path
org/jfrog/test

Deploy

Deploying an Artifact Bundle

An artifact bundle is deployed as a set of artifacts packaged in an archive with one of the following supported extensions: zip, tar, tar.gz, tgz.

When you specify that an artifact should be deployed as a bundle, Artifactory will extract the archive contents when you deploy it.

File structure within the archive

Artifacts should be packaged within the archive in the same file structure with which they should be deployed to the target repository.

To deploy an artifact bundle, in the **Deploy** dialog, first upload the archive file you want to deploy.

Check the **Deploy as Bundle Artifact** checkbox and click **Deploy**.

Deploy

✕

Target Repository

libs-release-local

Package Type: Maven

Type: **Single** | Multi

Upload Completed

Target Path ?

/bootstrap-3.3.5-dist.zip

Deploy as Bundle Artifact

Deploy

Deploying to a Virtual Repository

From version 4.2, Artifactory supports deploying artifacts to a virtual repository.

To enable this, you first need to designate one of the local repositories that is aggregated by the virtual repository as a deployment target. This can be done through the UI by setting the **Default Deployment Repository** in the **Basic Settings** of the **Edit Repository** screen.

The screenshot shows the 'Edit docker-virtual Repository' interface. The 'Basic' tab is selected, and the 'Repositories' section is expanded. The 'Default Deployment Repository' field is highlighted with a red box and contains 'docker-local'. The 'Selected Repositories' list includes 'docker-local' and 'docker-remote'. The 'Included Repositories' list also includes 'docker-local' and 'docker-remote'. The 'Docker Settings' section has 'Force Authentication' unchecked.

You can also set the Default Deployment Repository using the `defaultDeploymentRepo` parameter of the [Virtual Repository Configuration JSON](#) used in the [Create or Replace Repository Configuration](#) and [Update Repository Configuration](#) REST API endpoints. Once the deployment target is configured, you may deploy artifacts to it using any packaging format client configured to work with Artifactory. For example, `docker push`, `npm publish`, `NuGet push`, `gem push` etc.

You can also use Artifactory's REST API to [deploy an artifact](#) and use the virtual repository key in the path to deploy.

From version 4.4, if you do specify a **Default Deployment Repository** for a virtual repository, the corresponding [Set Me Up](#) dialog for the repository will also include instructions and code snippets for deploying to that repository.

Failed Uploads

The most common reasons for a rejected deployment are:

- Lack of permissions
- A conflict with the target repository's includes/excludes patterns
- A conflict with the target repository's snapshots/releases handling policy.

Manipulating Artifacts

Overview

Artifactory supports move, copy and deletion of artifacts to keep your repositories consistent and coherent. When an artifact is moved, copied or deleted, Artifactory immediately and automatically updates the corresponding metadata descriptors (such as *maven-metadata.xml*, RubyGems, Npm and more) to reflect the change and keep your repositories consistent with the package clients.

In addition, as a convenience feature, Artifactory provides a simple way to do a complete [version cleanup](#).

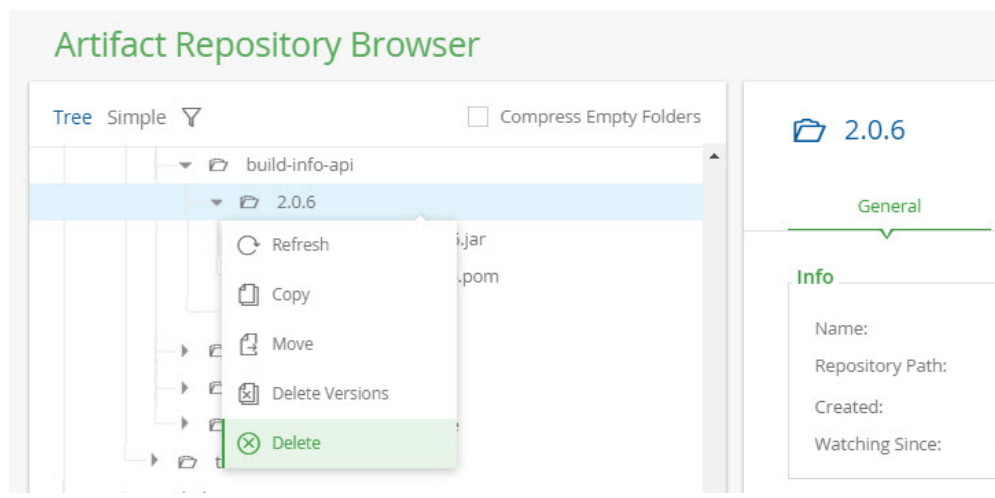
Page Contents

- [Overview](#)
- [Deleting a Single Item](#)
- [Deleting a Version](#)
- [Moving and Copying Artifacts](#)
 - [Simulating a Move or Copy](#)
- [Downloading a Folder](#)
 - [Configuring Folder Download](#)

Deleting a Single Item

To delete a single artifact or directory, select the item in the Tree Browser, and click **Delete** from the **Actions** menu or the right-click menu.

Once the item is deleted, the corresponding metadata file is updated to reflect the change so the item will not be found in a search.



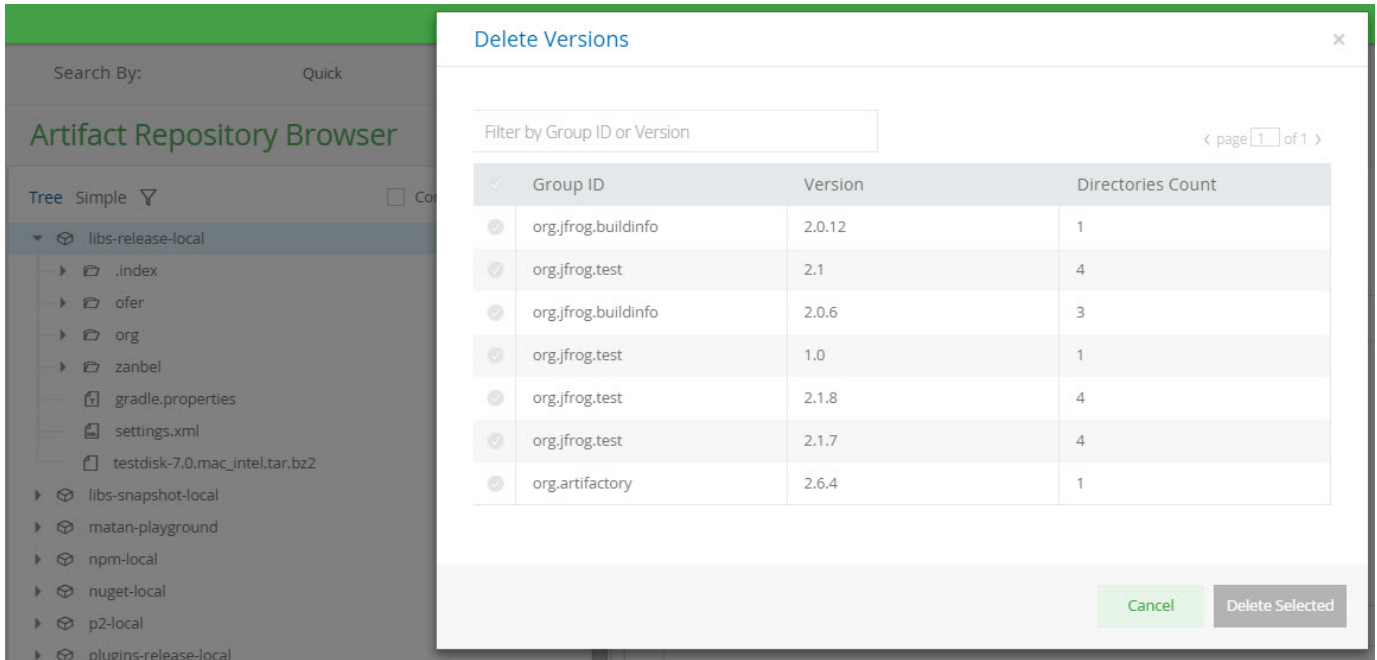
Deleting a Version

It is common for a repository to accumulate many different artifacts deployed under the same group (or path prefix) and the same version. This is

especially true for snapshot deployments of multi-module projects, where all deployed artifacts use the same version. To delete a version by individually deleting its constituent artifacts can be tedious, and would normally be managed by writing a dedicated script. Artifactory lets you select one of the artifacts in a version and then delete all artifacts with the same version tag in a single click while keeping the corresponding meta data descriptors up to date.

To delete a version, right-click a folder in the [Tree Browser](#) and select **Delete Versions...**

Artifactory drills down into the selected folder and returns with a list of all the groups and versions that can be deleted.



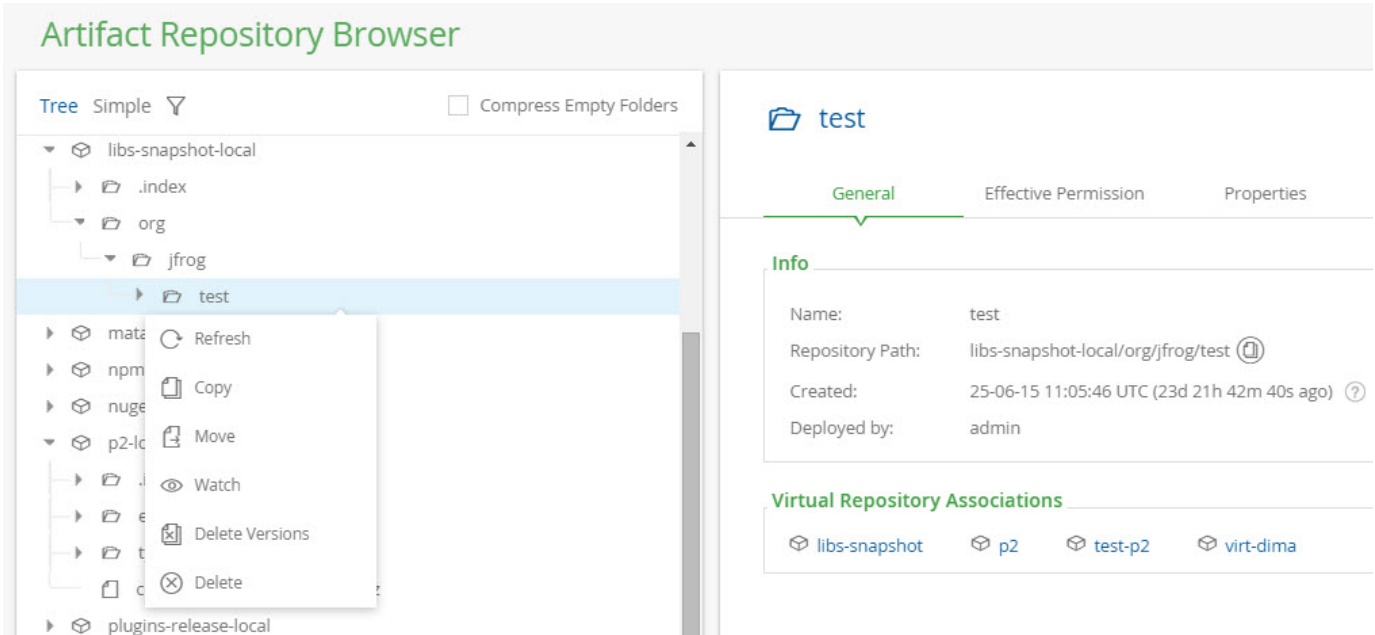
Select the versions you want to clean up and click **Delete Selected**

Limit to number of versions displayed

To avoid an excessively long search, Artifactory only displays the different version numbers assigned to the first 1000 artifacts found in the selected level of the repository hierarchy. If you do not see the version number you wish to delete, filter the artifacts displayed in the **Delete Versions** dialog by Group ID or Version number.

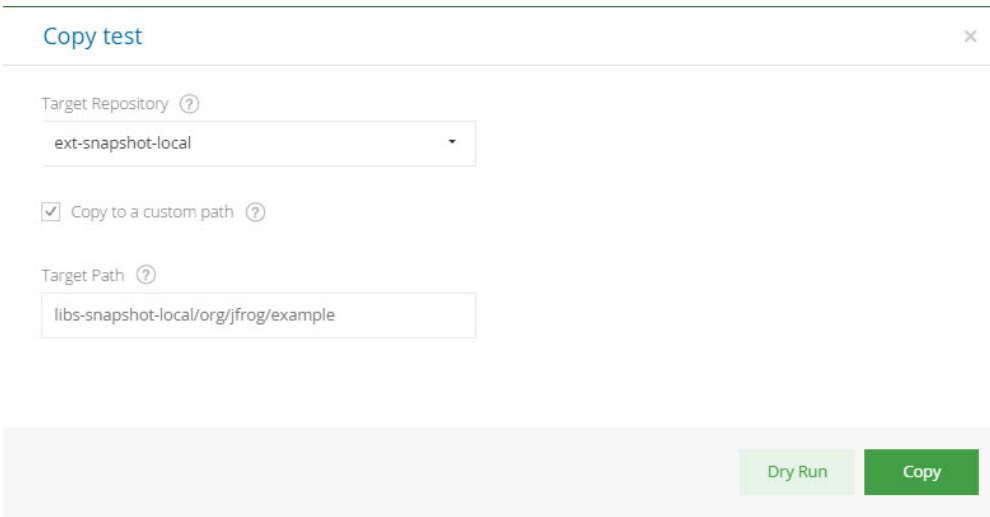
Moving and Copying Artifacts

To move or copy an artifact or folder, select it in the Tree Browser and then click **Move...** or **Copy...** from the **Actions** menu or from the right-click menu.



Artifactory will display a list of repositories from which you need to select your **Target Repository** for the operation.

The list of target repositories will contain all the local repositories, or virtual repositories with a "Default Deployment Repository" configured that are configured with the same package type as the source repository, or configured as generic repositories. This means, for example, you can only move an artifact from a debian repository to a generic repository or to a local debian repository.



After selecting your **Target Repository**, you can specify a custom **Target Path** if you want your source artifacts moved to a different location the **Target Repository**.

Copy operations are executed using Unix conventions

Copy operations are executed using Unix conventions. For example, copying *org/jfrog/1* from a source repository to *org/jfrog/1* in a target repository will result in the contents of the source being copied to *org/jfrog/1/1*.

To achieve the same path in the target repository, copy the source into one folder up in the hierarchy. In the above example, that would mean copying source *org/jfrog/1* into target *org/jfrog*.

If you leave the Target Path empty, the source will be copied into the target repository's root folder.

Custom target path suppresses cross-layout translation

If you are copying or moving your source artifacts to a repository with a different layout, specifying a **Custom Target Path** suppresses cross-layout translation. This means that your client may NOT be able to resolve the artifacts, even in cases of a same-layout operation.

Once you have selected your **Target Repository** (and **Custom Target Path** if needed), click **Move...** or **Copy...** to complete the operation.

All metadata is updated to reflect the operation once completed.

Simulating a Move or Copy

Note that an operation may fail or generate warnings for several reasons. For example, the target repository may not accept all the items due to its own specific policies, or you may not have the required permissions to perform the operation.

Before actually doing an operation, we recommend that you check if it will succeed without errors or warnings by clicking **Dry Run**.

Artifactory will run a simulation and display any errors and warnings that would appear when doing the actual operation.

Copy test ✕

Target Repository ?
ext-snapshot-local

Copy to a custom path ?

Target Path ?
libs-snapshot-local/org/jfrog/example

Dry Run Copy

Failed to validate target path of pom: libs-snapshot-local/org/jfrog/example/multi/2.18-SNAPSHOT/multi-2.18-20150625.110536-1.pom
Failed to validate target path of pom: libs-snapshot-local/org/jfrog/example/multi/2.18-SNAPSHOT/multi-2.18-

Permissions required

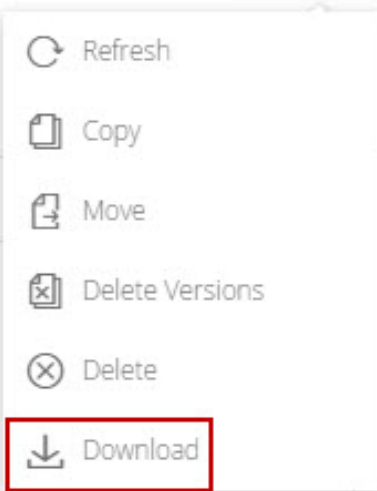
To successfully complete a move, you need to have DELETE permission on your source repository, and DEPLOY permission on your target repository

Downloading a Folder

Artifactory allows the download of a complete folder that is selected in the [Tree Browser](#) or [Simple Browser](#).

This ability is configurable by an Artifactory administrator, and if allowed, when a folder is selected the **Download** function is available in the **Actions** menu.

Actions



Download Folder

Folder Name: org/jfrog
Files Count: 1
Total Size: 0.44 MB

Archive Type
zip

Include Checksum files

[Download](#)

Archive Type	The Archive Type. Currently, zip, tar, tar.gz and tgz are supported.
Include Checksum Files	<p>Include SHA1, SHA256 and MD5 files - In Artifactory, checksum files (.sha1, .sha256 and .md5 files) are displayed and are downloadable in the HTML browsing endpoint (for example, <a href="http://<ARTIFACTORY>/artifactory/<REPOSITORY_KEY>">http://<ARTIFACTORY>/artifactory/<REPOSITORY_KEY>), depending on one of the below pre-conditions:</p> <ol style="list-style-type: none">1.The artifact was originally uploaded with its checksum value (i.e the deploying client provided a checksum header such as the "X-Checksum-Sha1" header on the request).2.The repository Checksum Policy is set to "Trust Server Generated Checksums". <p>If the latter applies, there is no need to provide the artifact checksums during the upload in order for its checksum files to be visible.</p> <p>The Download Folder functionality mimics this mechanism, and will write checksum files to the output archive based on the same conditions.</p> <p>*With remote repository caches, there is no distinction for the checksum policy of the repository. Simply checking this checkbox will always result in checksum files being added.</p>

You can also download a folder using the [Rest API](#).

Configuring Folder Download

An Artifactory administrator can enable complete folder download in the **Admin** module under **Admin | General Configuration**. This configuration will apply to all Artifactory users.

Folder Download Settings

- Enable Folder Download
- Enable Folder Download For Anonymous Access

Max Size [?](#)

Max Number of Files [?](#)

Max Parallel Folder Downloads [?](#)

Max Size	The maximum size (MB) of artifacts that can be downloaded in a folder download.
Max Number of Files	The maximum number of artifacts that may be downloaded from the selected folder and all its sub-folders.
Max Parallel Folder Downloads	The maximum number of concurrent folder downloads allowed.

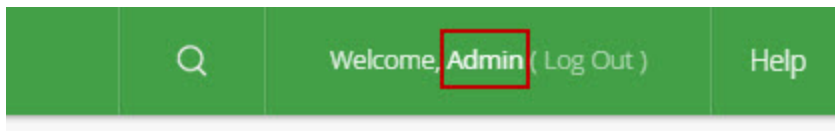
Updating Your Profile

Overview

Your profile page is used to manage the following aspects of your user profile:

- [API Key](#)
- [Password](#)
- [Email](#)
- [Bintray Settings](#)
- [Binding OAuth Accounts](#)

To display your profile page, click your login name on the top right-hand corner of the screen.



Unlocking Your Profile

To edit your profile, you first need to unlock it by entering your current password and clicking **Unlock**.

Once unlocked, you can modify all the elements of your user profile.

Page Contents

- Overview
- Unlocking Your Profile
- Changing your Personal Settings
 - API Key
 - Revoking or Regenerating an API Key
 - REST API
 - Changing Your Password and Email
 - Password Reminder
- Bintray Settings
- SSH Key
- Binding OAuth Accounts

Saving your changes

Be sure to click **Update** to save any changes to your profile.

Using external authentication

If you are using an external authentication server (such as [HTTP SSO](#), [OAuth SSO](#), or [SAML SSO](#)), you can ask your administrator to give you access to your [API key](#), [Bintray credentials](#) and [SSH public key](#) without having to unlock your profile.

Changing your Personal Settings

Personal settings include your Artifactory API Key, password and email address.

You are not able to change your password if Artifactory is configured to use external authentication such as LDAP.

API Key

Artifactory allows authentication for REST API calls using your API key as an alternative to your username and password in two ways: you may either use the `X-JFrog-Art-API` header with which you can specify an API key, or you may use basic authentication using your username and API key (instead of your password). For more details, please refer to the [REST API documentation](#).

Artifactory version

To use your API key for authentication, it must be generated using Artifactory 4.4.3 or later. If generate before, you **must** regenerate your API key and use the new key as a password for basic authentication.

Creating an API Key

To create an API Key, once you have unlocked your profile, click the "Generate" button next to the **API Key** field.

Authentication Settings

API Key

Click on Generate to create Key



Generate

Revoking or Regenerating an API Key

Once an API Key is created, it displayed, masked, in the corresponding field. Click the "View" icon to see the API Key in clear-text, or the "Copy" icon to copy the API Key to the clipboard.

To revoke the current API Key, click "Revoke API Key" Note that any REST API calls using the current API key for authentication will no longer be valid.

You may revoke the current API Key and create a new one in a single action by clicking "Regenerate". Any REST API calls using the current API key for authentication will no longer be valid, until you replace the API Key with the new one you just generated.

Authentication Settings

API Key

.....



⊗ Revoke API Key

Regenerate

REST API

The following REST API endpoints are available with regard to API Keys:

Endpoint	Description
Create API Key	Create an API key for the current user.
Get API Key	Get the current user's own API key.
Revoke API Key	Revokes the current user's API key.
Revoke User API Key	Revokes the API key of another user (requires Admin privileges).
Revoke All API Keys	Revokes all API keys currently defined in the system (requires Admin privileges).

Changing Your Password and Email

Once your profile is unlocked, Artifactory displays your password in an encrypted format that can be used whenever you need to provide your password in an environment that is not secure. For example, when making REST API calls over HTTP.

The encrypted password is initially masked, but you may click the "View" icon to view the encrypted password in clear-text. You may also click the "Copy" icon to copy the encrypted password to the clipboard.

To change your Artifactory password, enter your new password and verify it.

You can also modify your email address.

Personal Settings

New Password Password Strength

Retype Password

Email Address *

Authentication Settings

API Key 👁️ 🗑️ ⚙️

⊗ Revoke API Key

Encrypted Password 👁️ 🗑️

For more information about using secured passwords with your profile, please refer to [Centrally Secure Passwords](#).

Password Reminder

If you forget your password, on the Artifactory Login dialog, select **Forgot Password**, and enter your username in the following dialog that is displayed.

Welcome to JFrog Artifactory! ✕

Remember me

Forgot Password? Log In

Forgot Password ✕

Submit

When you click **Submit**, Artifactory will send a message to the email address configured for your user account, with a link you can click on to reset your password.

Bintray Settings

Bintray is JFrog's platform to store and distribute your software libraries. For more details please refer to the [JFrog Bintray Page](#).

Upon installation, Artifactory has Bintray's JCenter Java repository defined as a remote repository.

You may freely read from JCenter, and other Bintray repositories, however to upload an artifact, or perform other operations on Bintray through Artifactory such as search, you must have a Bintray account and provide your credentials through your profile page.

To provide your Bintray credentials, enter your **Bintray Username** and the **Bintray API Key** you received when registering to Bintray into the specified fields in Artifactory.

To verify that your credentials are valid you can click **Test**.

Bintray Settings

Bintray Username

Bintray API Key



[Register to Bintray...](#)

Test

SSH Key

From version 4.4, Artifactory supports authentication via SSH for the [Git LFS](#) client and the [Artifactory CLI](#).

To be authenticated via SSH, you need to enter your SSH public key in the **SSH Public Key (RSA)** field.



SSH

SSH Public Key (RSA)

```
Starts with 'ssh-rsa', 'ssh-dss', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', or 'ecdsa-sha2-nistp521'
```


Binding OAuth Accounts

From version 4.2, Artifactory is integrated with OAuth allowing you to log in through your account with one of the configured OAuth providers. To do so, you need to bind your OAuth account to your Artifactory user by clicking **Click to bind** next to the corresponding OAuth provider. For more details, please refer to [OAuth Integration](#).

OAuth User Binding	
 Git Enterprise	Click to bind
 Google-JFrog	Click to bind
 OpenID	Click to bind

Authentication

Overview

The following sections describe all the means of authentication available in Artifactory.

Basic Authentication

Artifactory provides a detailed and flexible permission-based system to control users' access to different features and artifacts.

To learn more, please refer to [Configuring Security](#).

LDAP

Artifactory supports authenticating users against an LDAP server out-of-the-box. When LDAP authentication is active, Artifactory first attempts to authenticate the user against the LDAP server. If LDAP authentication fails, Artifactory tries to authenticate via its internal database. For every LDAP authenticated user Artifactory creates a new user in the internal database (provided that the user does not already exist), and automatically assigns that user to the default groups.

To learn more, please refer to [Managing Security with LDAP](#).

Page Contents

- [Overview](#)
- [Basic Authentication](#)
- [LDAP](#)
- [Active Directory](#)
- [Single Sign-On](#)
- [SAML](#)
- [OAuth](#)
- [SSH](#)
- [Atlassian Crowd Integration](#)
- [Access Tokens](#)
- [Custom Authentication with User Plugins](#)

Active Directory

Artifactory supports integration with an Active Directory server to authenticate users and synchronize groups. When authentication using Active Directory is configured and active, Artifactory first attempts to authenticate the user against the Active Directory server. If the authentication fails,

Artifactory tries to authenticate via its internal database. For every externally authenticated user configured in your Active Directory server, Artifactory creates a new user in the internal database (provided the user does not already exist), and automatically assigns that user to the default groups.

To learn more, please refer to [Managing Security with Active Directory](#).

Single Sign-On

The Single Sign-on (SSO) Add-on allows you to reuse existing HTTP-based SSO infrastructures with Artifactory, such as the SSO modules offered by Apache HTTPd. Artifactory's authentication will work with commonly available SSO solutions, such as native NTLM, Kerberos, etc... SSO works by letting Artifactory know what trusted information it should look for in the HTTP request, assuming that this request has already been authenticated by the SSO infrastructure, which sits in front of Artifactory.

To learn more, please refer to [Single Sign-on](#).

SAML

SAML is an XML standard that allows you to exchange user authentication and authorization information between web domains. JFrog's Artifactory offers a SAML-based Single Sign-On service allowing federated Artifactory partners (identity providers) full control over the authorization process. Using SAML, Artifactory acts as service provider which receives users authentication information from external identity providers. In such case Artifactory is no longer responsible to authenticate the user although it still has to redirect the login request to the identity provider and verify the integrity of the identity provider's response.

To learn more, please refer to [SAML SSO Integration](#).

OAuth

OAuth integration allows you to delegate authentication requests to external providers and let users login to Artifactory using their accounts with those providers. Currently, Google, OpenID Connect, GitHub Enterprise and Cloud Foundry UAA are supported.

To learn more, please refer to [OAuth Integration](#).

SSH

Artifactory supports SSH authentication for Git LFS and the JFrog CLI using RSA public and private keys. SSH has the benefit of two-way authentication. In other words, before any sensitive data is exchanged between Artifactory and the client, the Artifactory server is authenticated to the client, and then the user operating Git LFS or JFrog CLI client is authenticated to Artifactory.

To learn more, please refer to [SSH Integration](#).

Atlassian Crowd Integration

The Atlassian Crowd Integration allows you to delegate authentication requests to Atlassian Crowd, use authenticated Crowd users and have Artifactory participate in a transparent SSO environment managed by Crowd. In addition, Atlassian Crowd Integration allows the use of JIRA User Server as an authentication server, but without support of SSO.

To learn more, please refer to [Atlassian Crowd and JIRA Integration](#).

Access Tokens

Artifactory offers the option for authentication through access tokens. An access token may be assigned to a user, or to an entity that is not an Artifactory user such as a job in a CI server. Permissions are assigned to access tokens by including them in Groups. Access tokens offer advantages such as cross-site authentication, limited-time access, authenticated access for non-users and more.

To learn more, please refer to [Access Tokens](#).

Custom Authentication with User Plugins

You can use User Plugins to implement custom authentication policies.

To learn more, please refer to [Management of Security Realms](#).

Artifactory REST API

Overview

Artifactory exposes its REST API through an auto-generated WADL file (courtesy of the [Jersey REST framework](#)).

This provides a convenient and up-to-date self-descriptive API and can be used by various tools/frameworks to automate the creation of REST calls.

The WADL file is available at the following URL:

```
http://server:port/artifactory/api/application.wadl
```

Usage

Artifactory REST API endpoints can be invoked in any of the standard ways to invoke a RESTful API. This section describes how to use the Artifactory REST API using cURL as an example.

Using and Configuring cURL

You can download cURL [here](#). Learn how to use and configure cURL [here](#).

Authentication

Artifactory's REST API supports four forms of authentication:

- Basic authentication using your username and password
- Basic authentication using your username and [API Key](#).
- Using a dedicated header (`X-JFrog-Art-API`) with your API Key.
- Using an [access token](#) instead of a password for basic authentication.
- Using an [access token](#) as a bearer token in an authorization header (`Authorization: Bearer`) with your access token.

Artifactory version

To use your API key for **Basic Authentication**, it must be generated using Artifactory 4.4.3 or later. If generated on a previous version, you **must** regenerate your API key and use the new key as a password for basic authentication.

Using JFrog CLI

JFrog CLI is a compact and smart client that provides a simple interface to automate access to Artifactory. As a wrapper to the REST API, it offers a way to simplify automation scripts making them more readable and easier to maintain, features such as parallel uploads and downloads, checksum optimization and wildcards/regular expressions make your scripts more efficient and reliable. Please note that several of the functions available through the REST API are also available through JFrog CLI and you should consider which method best meets your needs.

For more details on download, installation and usage of JFrog CLI, please refer to the [JFrog CLI User Guide](#).

Example - Deploying an Artifact

The example below demonstrates how to invoke the [Deploy Artifact REST API](#).

- You are using cURL from the unix command line, and are presently working from the **home (~) directory** of the user **'myUser'**
- You wish to deploy the file `'myNewFile.txt'`, which is located in your Desktop directory, (`'~/Desktop/myNewFile.txt'`)
- You have Artifactory running on your local system, on **port 8081**
- You wish to deploy the artifact into the `'my-repository'` repository, under the `'my/new/artifact/directory/'` folder structure, and wish to store the file there as `'file.txt'`
- You have configured a user in Artifactory named `'myUser'`, with password `'myP455w0rd!'`, and this

- user has permissions to deploy artifacts
- Your API Key is 'ABcdEF'
- Where possible, the same example is demonstrated using JFrog CLI

To deploy the file using your **username and password** for authentication, you would use the following command:

Using cURL with the REST API

```
curl -u myUser:myP455w0rd! -X PUT
"http://localhost:8081/artifactory/my-repository/my/new/artifact/directory/file.txt" -T
Desktop/myNewFile.txt
```

Using JFrog CLI

```
jfrog rt u file.txt
my-repository/my/new/artifact/directory/
--user=myUser --password=myP455w0rd!
```

To deploy the file using your **API Key** for basic authentication, you would use the following command:

Using cURL with the REST API

```
curl -u myUser:ABcdEF -X PUT
"http://localhost:8081/artifactory/my-repository/my/new/artifact/directory/file.txt" -T
Desktop/myNewFile.txt
```

Using JFrog CLI

```
jfrog rt u file.txt
my-repository/my/new/artifact/directory/
--apiKey=ABcdEF
```

To deploy the file using your **API Key** in a header, you would use the following command:

Using cURL with the REST API

```
curl -H "X-JFrog-Art-Api:ABcdEF" -X PUT
"http://localhost:8081/artifactory/my-repository/my/new/artifact/directory/file.txt" -T
Desktop/myNewFile.txt
```

To deploy the file using your **access token** for basic authentication, you would use the following command:

Using cURL with an access token

```
curl -u myUser:<Token> -X PUT
"http://localhost:8081/artifactory/my-repository/my/new/artifact/directory/file.txt" -T
Desktop/myNewFile.txt
```

To deploy the file using your **access token** in a header, you would use the following command:

Using cURL with an access token

```
curl -H "Authorization: Bearer <Token>" -X PUT
"http://localhost:8081/artifactory/my-repository/my/new/artifact/directory/file.txt" -T
Desktop/myNewFile.txt
```

Working with Artifactory SaaS

JFrog Artifactory SaaS offers the same extensive functionality and capabilities for automation as an on-prem installation, including authentication, use of JFrog CLI and the REST API endpoints.

As a SaaS service, the URL is different from an on-prem installation and the REST API endpoints can be reached at:

```
http://<Server Name>.jfrog.io/<Server Name>
```

Example

The snippets below apply the same [example](#) described above to an Artifactory SaaS instance named "myArtifactorySaas" (instead of to an on-prem installation).

To deploy the file using your **username and password** for authentication, you would use the following command:

Using cURL with the REST API for Artifactory SaaS

```
curl -u myUser:myP455w0rd! -X PUT
"http://myartifactorysaas.jfrog.io/myartifactorysaas/my-repository/my/new/artifact/directory/file.txt" -T
Desktop/myNewFile.txt
```

Using JFrog CLI

```
jfrog rt u file.txt
my-repository/my/new/artifact/directory/
--user=myUser --password=myP455w0rd!
```

Note that using JFrog CLI is identical with an Artifactory SaaS instance. You only need to [configure JFrog CLI](#) with the correct URL for your instance.

REST Resources

The sections below provide a comprehensive listing of the REST resources exposed by Artifactory.

For details on handling errors please refer to [ERROR RESPONSES](#) below.

Usage of REST resources is subject to security restrictions applicable to each individual resource.

BUILDS

All Builds

Description: Provides information on all builds

Since: 2.2.0

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/build

Produces: application/vnd.org.jfrog.build.Builds+json

Sample Output:

```
GET /api/build
{
  "uri":
  "http://localhost:8081/artifactory/api/build"
  "builds" : [
    {
      "uri" : "/my-build",
      "lastStarted" : ISO8601
      (YYYY-MM-dd'T'HH:mm:ss.SSSZ)
    },
    {
      "uri" : "/jackrabbit",
      "lastStarted" : ISO8601
      (YYYY-MM-dd'T'HH:mm:ss.SSSZ)
    }
  ]
}
```

Build Runs

Description: Build Runs

Since: 2.2.0

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/build/{buildName}

Produces: application/vnd.org.jfrog.build.BuildsByName+json

Sample Output:

```
GET /api/build/my-build
{
  "uri":
  "http://localhost:8081/artifactory/api/build/my-build
"
  "buildsNumbers" : [
    {
      "uri" : "/51",
      "started" : ISO8601
      (yyyy-MM-dd'T'HH:mm:ss.SSSZ)
    },
    {
      "uri" : "/52",
      "started" : ISO8601
      (yyyy-MM-dd'T'HH:mm:ss.SSSZ)
    }
  ]
}
```

Build Upload

Description: Upload Build

Security: Requires a privileged user (can be anonymous)

Notes: All build modules must have the `build.name` and `build.number` properties set as well as the correct SHA1 and MD5 in order to be properly linked in the build info.

Usage: `PUT /api/build/ -H "Content-Type: application/json" --upload-file build.json`

Consumes: `application/vnd.org.jfrog.build.BuildsByName+json`

Example: `curl -X PUT "http://localhost:8081/artifactory/api/build" -H "Content-Type: application/json" --upload-file build.json`

Sample format:

▼ [Click to view sample build.json](#)

```
{
  "properties" : {
    /* Environment variables and properties
    collected from the CI server.
    The "buildInfo.env." prefix is added to
    environment variables and build related properties.
    For system variables there's no prefix. */
    "buildInfo.env.JAVA_HOME" : "",
    ...
  },
  "version" : "1.0.1", // Build Info schema version
  "name" : "My-build-name", // Build name
  "number" : "28", // Build number
  "type" : "MAVEN", // Build type (values currently
  supported: MAVEN, GRADLE, ANT, IVY and GENERIC)
  "buildAgent" : { // Build tool information
    "name" : "Maven", // Build tool type
    "version" : "3.0.5" // Build tool version
```

```
    },
    "agent" : { // CI Server information
        "name" : "Jenkins", // CI server type
        "version" : "1.554.2" // CI server version
    },
    "started" : "2014-09-30T12:00:19.893+0300", //
Build start time in the format of
YYYY-MM-dd'T'HH:mm:ss.SSSZ
    "artifactoryPluginVersion" : "2.3.1",
    "durationMillis" : 9762, // Build duration in
milliseconds
    "artifactoryPrincipal" : "james", // Artifactory
principal (the Artifactory user used for
deployment)
    "url" :
"http://my-ci-server/jenkins/job/My-project-name/2
8/", // CI server URL
    "vcsRevision" :
"e4ab2e493afd369ae7bdc90d69c912e8346a3463", // VCS
revision
    "vcsUrl" :
"https://github.com/github-user/my-project.git", //
VCS URL
    "licenseControl" : { // Artifactory License
Control information
        "runChecks" : true, // Artifactory will run
automatic license scanning after the build is
complete (true/false)
        "includePublishedArtifacts" : true, // Should
Artifactory run license checks on the build
artifacts, in addition to the build dependencies
(true/false)
        "autoDiscover" : true, // Should Artifactory
auto discover licenses (true/false)
        "scopesList" : "", // A space-separated list of
dependency scopes/configurations to run license
violation checks on. If left empty all dependencies
from all scopes will be checked.
        "licenseViolationsRecipientsList" : "" //
Emails of recipients that should be notified of
license violations in the build info
(space-separated list)
    },
    "buildRetention" : { // Build retention
information
        "deleteBuildArtifacts" : true, // Automatically
remove build artifacts stored in Artifactory
(true/false)
        "count" : 100, // The maximum number of builds
to store in Artifactory.
        "minimumBuildDate" : 1407345768020, // Earliest
build date to store in Artifactory
        "buildNumbersNotToBeDiscarded" : [ ] // List of
```



```

build numbers that should not be removed from
Artifactory
  },
  /* List of build modules */
  "modules" : [ { // The build's first module
    "properties" : { // Module properties
      "project.build.sourceEncoding" : "UTF-8"
    },
    "id" : "org.jfrog.test:multi2:4.2-SNAPSHOT", //
Module ID
    /* List of module artifacts */
    "artifacts" : [ {
      "type" : "jar",
      "sha1" :
"2ed52ad1d864aec00d7a2394e99b3abca6d16639",
      "md5" : "844920070d81271b69579e17ddc6715c",
      "name" : "multi2-4.2-SNAPSHOT.jar"
    }, {
      "type" : "pom",
      "sha1" :
"e8e9c7d790191f4a3df3a82314ac590f45c86e31",
      "md5" : "1f027d857ff522286a82107be9e807cd",
      "name" : "multi2-4.2-SNAPSHOT.pom"
    } ],
    /* List of module dependencies */
    "dependencies" : [ {
      "type" : "jar",
      "sha1" :
"99129f16442844f6a4a11ae22fbbec40b14d774f",
      "md5" : "1f40fb782a4f2cf78f161d32670f7a3a",
      "id" : "junit:junit:3.8.1",
      "scopes" : [ "test" ]
    } ]
  }, { // The build's second module
    "properties" : { // Module properties
      "project.build.sourceEncoding" : "UTF-8"
    },
    "id" : "org.jfrog.test:multi3:4.2-SNAPSHOT", //
Module ID
    /* List of module artifacts */
    "artifacts" : [ { // Module artifacts
      "type" : "war",
      "sha1" :
"df8e7d7b94d5ec9db3bfc92e945c7ff4e2391c7c",
      "md5" : "423c24f4c6e259f2774921b9d874a649",
      "name" : "multi3-4.2-SNAPSHOT.war"
    }, {
      "type" : "pom",
      "sha1" :
"656330c5045130f214f954643fdc4b677f4cf7aa",
      "md5" : "b0afa67a9089b6f71b3c39043b18b2fc",
      "name" : "multi3-4.2-SNAPSHOT.pom"
    } ],

```

```

/* List of module dependencies */
"dependencies" : [ {
  "type" : "jar",
  "sha1" :
"a8762d07e76cfde2395257a5da47ba7c1dbd3dce",
  "md5" : "b6a50c8a15ece8753e37cbe5700bf84f",
  "id" : "commons-io:commons-io:1.4",
  "scopes" : [ "compile" ]
}, {
  "type" : "jar",
  "sha1" :
"342d1eb41a2bc7b52fa2e54e9872463fc86e2650",
  "md5" : "2a666534a425add50d017d4aa06a6fca",
  "id" :
"org.codehaus.plexus:plexus-utils:1.5.1",
  "scopes" : [ "compile" ]
}, {
  "type" : "jar",
  "sha1" :
"449ea46b27426eb846611a90b2fb8b4dcf271191",
  "md5" : "25c0752852205167af8f31aleb019975",
  "id" :
"org.springframework:spring-beans:2.5.6",
  "scopes" : [ "compile" ]
} ]
} ],
/* List of issues related to the build */
"issues" : {
  "tracker" : {
    "name" : "JIRA",
    "version" : "6.0.1"
  },
  "aggregateBuildIssues" : true, //whether or
not there are issues that already appeared in
previous builds
  "aggregationBuildStatus" : "Released",
  "affectedIssues" : [ {
    "key" : "RTFACT-1234",
    "url" :
"https://www.jfrog.com/jira/browse/RTFACT-1234",
    "summary" : "Description of the relevant
issue",
    "aggregated" : false //whether or not this
specific issue already appeared in previous builds
  }, {
    "key" : "RTFACT-5469",
    "url" :
"https://www.jfrog.com/jira/browse/RTFACT-5678",
    "summary" : "Description of the relevant
issue",
    "aggregated" : true
  } ]
} ]

```

```
}},  
}  
}
```

Build Info

Description: Build Info

Since: 2.2.0

Security: Requires a privileged user with deploy permissions (can be anonymous)

Usage: GET /api/build/{buildName}/{buildNumber}

Produces: application/vnd.org.jfrog.build.BuildInfo+json

Sample Output:

```
GET /api/build/my-build/51  
{  
  "uri":  
  "http://localhost:8081/artifactory/api/build/my-build  
/51"  
  "buildInfo" : {  
    ...  
  }  
}
```

Builds Diff

Description: Compare a build artifacts/dependencies/environment with an older build to see what has changed (new artifacts added, old dependencies deleted etc).

Since: 2.6.6

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/build/{buildName}/{buildNumber}?diff={OlderbuildNumber}

Produces: application/vnd.org.jfrog.build.BuildsDiff+json

Sample Output:

```
GET /api/build/my-build/51?diff=50
{
  "artifacts": {
    "updated": [],
    "unchanged": [],
    "removed": [],
    "new": []
  }, "dependencies": {
    "updated": [],
    "unchanged": [],
    "removed": [],
    "new": []
  }, "properties": {
    "updated": [],
    "unchanged": [],
    "removed": [],
    "new": []
  }
}
```

Page contents

- Overview
- Usage
 - Authentication
 - Using JFrog CLI
 - Example - Deploying an Artifact
 - Working with Artifactory SaaS
 - Example
- REST Resources
 - BUILDS
 - All Builds
 - Build Runs
 - Build Upload
 - Build Info
 - Builds Diff
 - Build Promotion
 - Promote Docker Image
 - Delete Builds
 - Build Rename
 - Push Build to Bintray
 - Distribute Build
 - Control Build Retention
 - ARTIFACTS & STORAGE
 - Folder Info
 - File Info
 - Get Storage Summary Info
 - Item Last Modified
 - File Statistics
 - Item Properties
 - Set Item Properties
 - Delete Item Properties
 - Set Item SHA256 Checksum
 - Retrieve Artifact
 - Retrieve Latest Artifact
 - Retrieve Build Artifacts Archive
 - Retrieve Folder or Repository Archive
 - Trace Artifact Retrieval
 - Archive Entry Download
 - Create Directory
 - Deploy Artifact

- Deploy Artifact by Checksum
- Deploy Artifacts from Archive
- Push a Set of Artifacts to Bintray
- Push Docker Tag to Bintray
- Distribute Artifact
- File Compliance Info
- Delete Item
- Copy Item
- Move Item
- Get Repository Replication Configuration
- Set Repository Replication Configuration
- Update Repository Replication Configuration
- Delete Repository Replication Configuration
- Scheduled Replication Status
- Pull/Push Replication
- Pull/Push Replication (Deprecated)
- Create or Replace Local Multi-push Replication
- Update Local Multi-push Replication
- Delete Local Multi-push Replication
- Enable or Disable Multiple Replications
- Get Global System Replication Configuration
- Block System Replication
- Unblock System Replication
- Artifact Sync Download
- Folder Sync (Deprecated)
- File List
- Get Background Tasks
- Empty Trash Can
- Delete Item From Trash Can
- Restore Item from Trash Can
- Optimize System Storage
- Get Puppet Modules
- Get Puppet Module
- Get Puppet Releases
- Get Puppet Release
- SEARCHES
 - Artifactory Query Language (AQL)
 - Artifact Search (Quick Search)
 - Archive Entries Search (Class Search)
 - GAVC Search
 - Property Search
 - Checksum Search
 - Bad Checksum Search
 - Artifacts Not Downloaded Since
 - Artifacts With Date in Date Range
 - Artifacts Created in Date Range
 - Pattern Search
 - Builds for Dependency
 - License Search
 - Artifact Version Search
 - Artifact Latest Version Search Based on Layout
 - Artifact Latest Version Search Based on Properties
 - Build Artifacts Search
 - List Docker Repositories
 - List Docker Tags
- SECURITY
 - Get Users
 - Get User Details
 - Get User Encrypted Password
 - Create or Replace User
 - Update User
 - Delete User
 - Expire Password for a Single User
 - Expire Password for Multiple Users
 - Expire Password for All Users
 - Unexpire Password for a Single User
 - Change Password
 - Get Password Expiration Policy
 - Set Password Expiration Policy
 - Configure User Lock Policy
 - Retrieve User Lock Policy
 - Get Locked Out Users

- Unlock Locked Out User
- Unlock Locked Out Users
- Unlock All Locked Out Users
- Create API Key
- Regenerate API Key
- Get API Key
- Revoke API Key
- Revoke User API Key
- Revoke All API Keys
- Get Groups
- Get Group Details
- Create or Replace Group
- Update Group
- Delete Group
- Get Permission Targets
- Get Permission Target Details
- Create or Replace Permission Target
- Delete Permission Target
- Effective Item Permissions
- Security Configuration
- Save Security Configuration (Deprecated)
- Activate Master Key Encryption
- Deactivate Master Key Encryption
- Set GPG Public Key
- Get GPG Public Key
- Set GPG Private Key
- Set GPG Pass Phrase
- Create Token
- Refresh Token
- Revoke Token
- Get Service ID
- Get Certificates
- Add Certificate
- Delete Certificate
- REPOSITORIES
 - Get Repositories
 - Repository Configuration
 - Create Repository
 - Update Repository Configuration
 - Delete Repository
 - Remote Repository Configuration (Deprecated)
 - Calculate YUM Repository Metadata
 - Calculate NuGet Repository Metadata
 - Calculate Npm Repository Metadata
 - Calculate Maven Index
 - Calculate Maven Metadata
 - Calculate Debian Repository Metadata
 - Calculate Opkg Repository Metadata
 - Calculate Bower Index
- SYSTEM & CONFIGURATION
 - System Info
 - System Health Ping
 - Verify Connection
 - General Configuration
 - Save General Configuration
 - Update Custom URL Base
 - License Information
 - Install License
 - HA License Information
 - Install HA Cluster Licenses
 - Delete HA Cluster License
 - Version and Add-ons information
 - Get Reverse Proxy Configuration
 - Update Reverse Proxy Configuration
 - Get Reverse Proxy Snippet
 - Create Bootstrap Bundle
- PLUGINS
 - Execute Plugin Code
 - Retrieve Plugin Code
 - Retrieve Plugin Info
 - Retrieve Plugin Info Of A Certain Type
 - Retrieve Build Staging Strategy

- Execute Build Promotion
- Reload Plugins
- IMPORT & EXPORT
 - Import Repository Content
 - Import System Settings Example
 - Full System Import
 - Export System Settings Example
 - Export System
- SUPPORT
 - Create Bundle
 - List Bundles
 - Get Bundle
 - Delete Bundle
- ERROR RESPONSES

Read More

- [Repository Configuration JSON](#)
- [Security Configuration JSON](#)
- [System Settings JSON](#)

Build Promotion

Description: Change the status of a build, optionally moving or copying the build's artifacts and its dependencies to a target repository and setting properties on promoted artifacts.

All artifacts from all scopes are included by default while dependencies are not. Scopes are additive (or).

Since: 2.3.3

Notes: Requires Artifactory Pro

Security: Requires a privileged user (can be anonymous)

Usage: POST /api/build/promote/{buildName}/{buildNumber}

Consumes: application/vnd.org.jfrog.artifactory.build.PromotionRequest+json

```

POST /api/build/promote/my-build/51
{
  "status": "staged",          // new build status (any string)
  "comment" : "Tested on all target platforms.", // An optional comment
  describing the reason for promotion. Default: ""
  "ciUser": "builder",        // The user that invoked promotion from the CI
  server
  "timestamp" : ISO8601,      // the time the promotion command was received
  by Artifactory (It needs to be unique),
  // the format is: 'yyyy-MM-dd'T'HH:mm:ss.SSSZ'. Example:
  '2016-02-11T18:30:24.825+0200'.
  "dryRun" : false,           // run without executing any operation in
  Artifactory, but get the results to check if the operation can succeed.
  Default: false
  "sourceRepo" : "libs-snapshot-local", // optional repository from which
  the build's artifacts will be copied/moved
  "targetRepo" : "libs-release-local", // optional repository to move or
  copy the build's artifacts and/or dependencies
  "copy": false,              // whether to copy instead of move, when a target
  repository is specified. Default: false
  "artifacts" : true,         // whether to move/copy the build's artifacts.
  Default: true
  "dependencies" : false,     // whether to move/copy the build's
  dependencies. Default: false.
  "scopes" : [ "compile", "runtime" ], // an array of dependency scopes
  to include when "dependencies" is true
  "properties": {             // a list of properties to attach to the build's
  artifacts (regardless if "targetRepo" is used).
    "components": ["c1","c3","c14"],
    "release-name": ["fb3-ga"]
  },
  "failFast": true           // fail and abort the operation upon receiving an
  error. Default: true
}

```

Produces: application/vnd.org.jfrog.artifactory.build.PromotionResult+json

Sample Output:

```

{
  "messages" : [
    {
      "level": "error",
      "message": "The repository has denied...."
    },...
  ]
}

```

Promote Docker Image

Description: Promotes a Docker image from one repository to another

Since: 3.7

Notes: Requires Artifactory Pro

Security: Requires a privileged user

Usage: POST api/docker/<repoKey>/v2/promote

Consumes: application/json

```
{
  "targetRepo" : "<targetRepo>",          // The target repository for the
move or copy
  "dockerRepository" : "<dockerRepository>",    // The docker repository
name to promote
  "targetDockerRepository" : "<targetDockerRepository>" // An optional
docker repository name, if null, will use the same name as
'dockerRepository'
  "tag" : "<tag>",                // An optional tag name to promote, if null -
the entire docker repository will be promoted. Available from v4.10.
  "targetTag" : "<tag>",          // An optional target
tag to assign the image after promotion, if null - will use the same tag
  "copy": false           // An optional value to set whether to copy
instead of move. Default: false
}
```

Produces: application/text

Sample Usage:

```
POST api/docker/docker-local/v2/promote
{
  "targetRepo": "docker-prod",
  "dockerRepository": "jfrog/ubuntu"
}
```

Delete Builds

Description: Removes builds stored in Artifactory. Useful for cleaning up old build info data.

If the `artifacts` parameter is evaluated as 1 (0/false by default), build artifacts are also removed provided they have the corresponding `build.name` and `build.number` properties attached to them.

If the `deleteAll` parameter is evaluated as 1 (0/false by default), the whole build is removed.

Since: 2.3.0; artifact removal since 2.3.3;

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: DELETE /api/build/{buildName}[?buildNumbers=n1[,n2]][&artifacts=0/1][&deleteAll=0/1]

Produces: application/text

Sample Usage:

```
DELETE /api/build/my-build?buildNumbers=51,52,55&artifacts=1
```

```
The following builds has been deleted successfully: 'my-build#51',
'my-build#52', 'my-build#55'.
```

```
DELETE /api/build/my-build?deleteAll=1
```

```
All 'my-build' builds have been deleted successfully.
```

Build Rename

Description: Renames a build stored in Artifactory. Typically used to keep the build info in sync with a renamed build on the CI server.

Since: 2.2.5

Notes: Requires Artifactory Pro

Security: Requires a valid user with deploy permissions

Usage: POST /api/build/rename/{buildName}?to=newBuildName

Produces: application/text

Sample Usage:

```
POST /api/build/rename/myJobName?to=myNewJobName
```

```
Build renaming of 'myJobName' to 'myNewJobName' was successfully started.
```

Push Build to Bintray

Deprecated: This endpoint is deprecated and is replaced with [Distribute Build](#)

Description: Push a build to Bintray as a version.

Uses a descriptor file (that must have 'bintray-info' in its filename and a .json extension) that is included with the build artifacts. For more details, please refer to [Pushing a Build](#).

Signing a version is controlled by the `gpgSign` parameter in the descriptor file, and the `gpgSign` parameter passed to this command. **The value passed to this command always takes precedence over the value in the descriptor file.**

If you also want a passphrase to be applied to your signature, specify `gpgPassphrase=<passphrase>`.

You may omit the descriptor file by passing 6 override parameters (see below). If you wish to use the descriptor file you should pass an empty json string instead.

Since: 3.5.0

Security: Requires a valid user with deploy permissions and Bintray credentials defined (for more details, please refer to [Entering your Bintray credentials](#)).

Usage: POST /api/build/pushToBintray/{build.name}/{build.number}?gpgPassphrase=<passphrase>[&gpgSign=true|false]

Consumes: application/vnd.org.jfrog.artifactory.build.BintrayDescriptorOverrideParams+json

Sample Input:

```
POST
/api/build/pushToBintray/testBuild/1?gpgPassphrase=password&gpgSign=true
{
  "subject": "myUser",
  "repoName": "test",
  "packageName": "overridePkg",
  "versionName": "overrideVer",
  "licenses": ["MIT"],
  "vcs_url": "https://github.com/bintray/bintray-client-java"
}
```

Produces: application/vnd.org.jfrog.artifactory.bintray.BintrayPushResponse+json

Sample Output:

```
{"Message": "Pushing build to Bintray finished successfully."}
```

Distribute Build

Description: Deploys builds from Artifactory to Bintray, and creates an entry in the corresponding Artifactory distribution repository specified.

Notes: Requires Artifactory Pro

Since: 4.8

Security: Requires an authenticated user.

Usage: POST /api/build/distribute/{buildName}/{buildNumber}

Consumes: application/json

```
{
  "publish" : "<true | false>"      // Default: true. If true, builds are
  published when deployed to Bintray
  "overrideExistingFiles" : "<true | false>" // Default: false. If true,
  Artifactory overwrites builds already existing in the target path in
  Bintray.
      // Existing version attributes are also overridden if defined
  in the distribution repository Advanced Configuration
  "pgpPassphrase" : "<passphrase>"    // If specified, Artifactory will GPG
  sign the build deployed to Bintray and apply the specified passphrase
  "async" : "<true | false>"          // Default: false. If true, the build will
  be distributed asynchronously. Errors and warnings may be viewed in the log
  file
  "targetRepo" : "<targetDistributionRepo>", // The Distribution Repository
  into which artifacts should be deployed
  "sourceRepos" : ["<repoKey>"]        // An array of local repositories from
  which build artifacts should be deployed
  "dryRun" : "<true | false>"          // Default: false. If true, distribution
  is only simulated. No files are actually moved.
}
```

Sample input:

```
POST /api/build/distribute/my-build/1
{
  "targetRepo" : "dist-repo-jfrog-artifactory",
  "sourceRepos" : ["yum-local"]
}
```

Control Build Retention

Description: Specifies retention parameters for build info

Since: 5.2.1

Security: Requires a privileged user with deploy permissions (can be anonymous)

Usage: POST /api/build/retention/{buildName}?async=<true | false>

Consumes: application/json

```
{
  "deleteBuildArtifacts" : <true | false>, // When true, automatically
  removes build artifacts stored in Artifactory
  "count" : <count>, // The maximum number of builds to store in
  Artifactory.
  "minimumBuildDate" : <date>, // Earliest build date to store in
  Artifactory - ISO8601 (yyyy-MM-dd'T'HH:mm:ss.SSSZ)
  "buildNumbersNotToBeDiscarded" : [ ] // List of build numbers that
  should not be removed from Artifactory
}
```

Sample Usage:

```
POST /api/build/retention/myBuild?async=true

{
  "deleteBuildArtifacts" : true,
  "count" : 100, //
  "minimumBuildDate" : 1407345768020,
  "buildNumbersNotToBeDiscarded" : [ 5, 9 ]
}
```

ARTIFACTS & STORAGE

Folder Info

Description: Folder Info

For virtual use, the virtual repository returns the unified children. Supported by local, local-cached and virtual repositories.

Since: 2.2.0

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/storage/{repoKey}/{folder-path}

Produces: application/vnd.org.jfrog.artifactory.storage.FolderInfo+json

Sample Output:

```
GET /api/storage/libs-release-local/org/acme
{
  "uri":
  "http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme",
  "repo": "libs-release-local",
  "path": "/org/acme",
  "created": ISO8601 (yyyy-MM-dd'T'HH:mm:ss.SSSZ),
  "createdBy": "userY",
  "lastModified": ISO8601 (yyyy-MM-dd'T'HH:mm:ss.SSSZ),
  "modifiedBy": "userX",
  "lastUpdated": ISO8601 (yyyy-MM-dd'T'HH:mm:ss.SSSZ),
  "children": [
    {
      "uri" : "/child1",
      "folder" : "true"
    }, {
      "uri" : "/child2",
      "folder" : "false"
    }
  ]
}
```

File Info

Description: File Info

For virtual use the virtual repository returns the resolved file. Supported by local, local-cached and virtual repositories.

Since: 2.2.0

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/storage/{repoKey}/{filePath}

Produces: application/vnd.org.jfrog.artifactory.storage.FileInfo+json

Sample Output:

```
GET /api/storage/libs-release-local/org/acme/lib/ver/lib-ver.pom
{
  "uri":
  "http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme
  /lib/ver/lib-ver.pom",
  "downloadUri":
  "http://localhost:8081/artifactory/libs-release-local/org/acme/lib/ver/lib
  -ver.pom",
  "repo": "libs-release-local",
  "path": "/org/acme/lib/ver/lib-ver.pom",
  "remoteUrl": "http://some-remote-repo/mvn/org/acme/lib/ver/lib-ver.pom",
  "created": ISO8601 (yyyy-MM-dd'T'HH:mm:ss.SSSZ),
  "createdBy": "userY",
  "lastModified": ISO8601 (yyyy-MM-dd'T'HH:mm:ss.SSSZ),
  "modifiedBy": "userX",
  "lastUpdated": ISO8601 (yyyy-MM-dd'T'HH:mm:ss.SSSZ),
  "size": "1024", //bytes
  "mimeType": "application/pom+xml",
  "checksums":
  {
    "md5" : string,
    "sha1" : string,
    "sha256" : string
  },
  "originalChecksums":{
    "md5" : string,
    "sha1" : string,
    "sha256" : string
  }
}
```

Get Storage Summary Info

Description: Returns storage summary information regarding binaries, file store and repositories.

Since: 4.2.0

Security: Requires a privileged user (Admin only)

Usage: GET /api/storageinfo

Produces: application/json

Sample Output:

```

GET /api/storageinfo
{
  "binariesSummary": {
    "binariesCount": "125,726",
    "binariesSize": "3.48 GB",
    "artifactsSize": "59.77 GB",
    "optimization": "5.82%",
    "itemsCount": "2,176,580",
    "artifactsCount": "2,084,408"
  },
  "fileStoreSummary": {
    "storageType": "filesystem",
    "storageDirectory": "/home/.../artifactory/devenv/.artifactory/data/filestore",
    "totalSpace": "204.28 GB",
    "usedSpace": "32.22 GB (15.77%)",
    "freeSpace": "172.06 GB (84.23%)"
  },
  "repositoriesSummaryList": [
    {
      "repoKey": "plugins-release",
      "repoType": "VIRTUAL",
      "foldersCount": 0,
      "filesCount": 0,
      "usedSpace": "0 bytes",
      "itemsCount": 0,
      "packageType": "Maven",
      "percentage": "0%"
    },
    {
      "repoKey": "repo",
      "repoType": "VIRTUAL",
      "foldersCount": 0,
      "filesCount": 0,
      "usedSpace": "0 bytes",
      "itemsCount": 0,
      "packageType": "Generic",
      "percentage": "0%"
    },
    ...
    {
      "repoKey": "TOTAL",
      "repoType": "NA",
      "foldersCount": 92172,
      "filesCount": 2084408,
      "usedSpace": "59.77 GB",
      "itemsCount": 2176580
    }
  ]
}

```

Item Last Modified

Description: Retrieve the last modified item at the given path. If the given path is a folder, the latest last modified item is searched for recursively. Supported by local and local-cached repositories.

Since: 2.2.5

Notes: Requires Artifactory Pro

Security: Requires a valid user with deploy permissions

Usage: GET /api/storage/{repoKey}/{item-path}?lastModified

Produces: application/vnd.org.jfrog.artifactory.storage.ItemLastModified+json

Sample Output:

```
GET /api/storage/libs-release-local/org/acme?lastModified
{
  "uri":
  "http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme
  /foo/1.0-SNAPSHOT/foo-1.0-SNAPSHOT.pom",
  "lastModified": ISO8601
}
```

File Statistics

Description: Item statistics record the number of times an item was downloaded, last download date and last downloader. Supported by local and local-cached repositories.

Since: 3.1.0

Security: Requires read privileges

Usage: GET /api/storage/{repoKey}/{item-path}?stats

Produces: application/vnd.org.jfrog.storage.StatsInfo+json

Sample Output:

```
GET /api/storage/libs-release-local/org/acme/foo/1.0/foo-1.0.jar?stats
{
  "uri":
  "http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme
  /foo/1.0/foo-1.0.jar",
  "lastDownloaded": Timestamp (ms),
  "downloadCount": 1337,
  "lastDownloadedBy": "user1"
}
```

Item Properties

Description: Item Properties. Optionally return only the properties requested.

Since: 2.2.1

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/storage/{repoKey}/{itemPath}?properties[=x[,y]]

Produces: application/vnd.org.jfrog.artifactory.storage.ItemProperties+json

Sample Output:

```
GET /api/storage/libs-release-local/org/acme?properties\[=x[,y]\]
{
  "uri":
  "http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme"
  "properties":{
    "p1": ["v1", "v2", "v3"],
    "p2": ["v4", "v5", "v6"]
  }
}
```

Set Item Properties

Description: Attach properties to an item (file or folder). When a folder is used property attachment is recursive by default.

In order to supply special characters (comma (,), backslash(\), pipe(|), equals(=)) as key/value you must add an encoded backslash (%5C) before them. For example: `..?properties=a=1%5C=1` will attach key a with 1=1 as value.

To specify multiple properties, you can separate the items in one of the following ways:

- Use a semicolon - ; (recommended)
- Use the encoding for the pipe ("|") character - %7C
Alternatively, you may configure your NGINX to encode URLs so that if an unencoded pipe is used in the URL, NGINX will encode it to %7C. We recommend that you verify that this configuration does not break any other systems served by NGINX

Supported by local and local-cached repositories.

Notes: Requires Artifactory Pro

The following special characters are forbidden in the key field:) ({ } [* + ^ \$ \ / ~ ` ! @ # % & < > ; = , ± § and the Space character.

Since: 2.3.0

Security: Requires a privileged user (can be anonymous)

Usage: PUT /api/storage/{repoKey}/{itemPath}?properties=p1=v1[,v2][|p2=v3][&recursive=1]

Sample Usage:

```
PUT
/api/storage/libs-release-local/ch/qos/logback/logback-classic/0.9.9?properties=os=win,linux;qa=done&recursive=1
```

Delete Item Properties

Description: Deletes the specified properties from an item (file or folder). When a folder is used property removal is recursive by default. Supported by local and local-cached repositories.

Notes: Requires Artifactory Pro

Since: 2.3.2

Security: Requires a privileged user (can be anonymous)

Usage: DELETE /api/storage/{repoKey}/{itemPath}?properties=p1[,p2][&recursive=1]

Sample Usage:

```
DELETE
/api/storage/libs-release-local/ch/qos/logback/logback-classic/0.9.9?properties=os,qa&recursive=1
```

Set Item SHA256 Checksum

Description: Calculates an artifact's SHA256 checksum and attaches it as a property (with key "sha256"). If the artifact is a folder, then recursively calculates the SHA256 of each item in the folder and attaches the property to each item.

Since: 4.2.1

Security: Requires an admin user

Consumes: application/json

Usage: POST /api/checksum/sha256 -H "Content-Type: application/json"

Sample Usage:

```
POST /api/checksum/sha256 -H "Content-Type: application/json"
{
  "repoKey": "ext-snapshot-local",
  "path": "artifactory-powerpack-3.9.3/bin/"
}
```

Retrieve Artifact

Description: Retrieves an artifact from the specified destination.

You can also use [Property-based Resolution](#) as part of retrieving artifacts.

Security: Requires a user with 'read' permission (can be anonymous)

Usage: GET /repo-key/path/to/artifact.ext

Sample Usage:

```
GET
http://localhost:8081/artifactory/libs-release-local/ch/qos/logback/logback-classic/0.9.9/logback-classic-0.9.9.jar
```

Retrieve Latest Artifact

Description: Retrieves the latest artifact version from the specified destination.

Latest Maven Release/Integration: Specify `SNAPSHOT` or `[RELEASE]` for the version in the requested path to get the latest Maven integration or release artifact.

Latest Non-Maven Release/Integration: Specify `[INTEGRATION]` or `[RELEASE]` for the version in the requested path (replacing the `[folderItegRev]` and `[fileItegRev]` as defined by the repository's [layout](#)) to get the latest integration version or latest release version artifact accordingly based on alphabetical sorting.

Integration and release tokens cannot be mixed together.

You can also use [property-based resolution](#) as part of retrieving artifacts to restrict resolution of artifacts assigned with specific properties.

NOTE:

1. Only local, cache and virtual repositories will be used.
2. To change the retrieve latest behavior to retrieve the latest version based on the created date you can add the following flag to `$ARTIFACTORY_HOME/etc/artifactory.system.properties` and add the following flag `artifactory.request.searchLatestReleaseByDateCreated=true` and restart Artifactory service.

Notes: Requires Artifactory Pro.

Since: Latest Maven: 2.6.0; Latest non-Maven: 2.6.2

Security: Requires a user with 'read' permission (can be anonymous)

Usage: GET /repo-key/path/to/artifact.ext

Sample Usage:

Download the latest Maven unique snapshot artifact:

```
GET
http://localhost:8081/artifactory/libs-release-local/ch/qos/logback/logback-classic/0.9.9-SNAPSHOT/logback-classic-0.9.9-SNAPSHOT.jar
```

Download the latest release artifact:

```
GET
http://localhost:8081/artifactory/ivy-local/org/acme/[RELEASE]/acme-[RELEASE].jar
```

Download the latest integration artifact:

```
GET
http://localhost:8081/artifactory/ivy-local/org/acme/1.0-[INTEGRATION]/acme-1.0-[INTEGRATION].jar
```

Retrieve Build Artifacts Archive

Description: Retrieves an archive file (supports zip/tar/tar.gz/tgz) containing all the artifacts related to a specific build, you can optionally provide mappings to filter the results, the mappings support [regex capturing groups](#) which enables you to dynamically construct the target path inside the result archive file.

Notes: Requires Artifactory Pro

Since: 2.6.5

Security: Requires a privileged user (can be anonymous)

Usage: POST /api/archive/buildArtifacts -H "Content-Type: application/json"

Consumes: application/vnd.org.jfrog.artifactory.build.BuildArtifactsRequest+json

Produces: application/zip (for zip archive type), application/x-tar (for tar archive type), application/x-gzip (for tar.gz/tgz archive type)

Sample Usage:

```
POST /api/archive/buildArtifacts -H "Content-Type: application/json"
{
  +"buildName": "build-name" // The build name for search by
  +"buildNumber": "15" // The build number to search by, can be LATEST to
search for the latest build number
  -"buildStatus": "Released" // Optionally search by latest build status
(e.g: "Released")
  -"repos": ["libs-release-local,ext-release-local"] // Optionally refine
search for specific repos, omit to search within all repositories
  +"archiveType": "tar/zip/tar.gz/tgz" // The archive file type to return
  -"mappings": [ // Optionally refine the search by providing a list of
regex patterns to search by
  {
    "input": "(.+)/(.+)-sources.jar",
    "output": "$1/sources/$2.jar" // Optionally provide different path of the
found artifacts inside the result archive, supports regex groups tokens
  },
  {
    "input": "(.)-release.zip"
  }
]
}
```

Retrieve Folder or Repository Archive

Description: Retrieves an archive file (supports zip/tar/tar.gz/tgz) containing all the artifacts that reside under the specified path (folder or repository root). Requires [Enable Folder Download](#) to be set.

Notes: Requires Artifactory Pro

Since: 4.1.0

Security: Requires a privileged user with read permissions on the path.

Usage: GET /api/archive/download/{repoKey}/{path}?archiveType={archiveType}&includeChecksumFiles=true]

Produces: */*

Sample Usage:

```
GET /api/archive/download/my-local-repo/path/to/folder?archiveType=zip
{Stream containing contents of path my-local-repo/path/to/folder}

GET /api/archive/download/my-local-repo?archiveType=zip
{Stream containing contents of repo my-local-repo}
```

Trace Artifact Retrieval

Description: Simulates an artifact retrieval request from the specified location and returns verbose output about the resolution process. This API is useful for debugging artifact retrieval issues.

Security: As applied to standard artifact retrieval by the requesting user.

Since: 2.6.0

Usage: GET /repo-key/path/to/artifact.ext?trace

Produces: text/plain

Sample Output:

```
GET
http://localhost:8081/artifactory/libs-release-local/jmock/jmock/1.0.1/jmock-1.0.1.jar?trace

Request ID: 51c808f6
Repo Path ID: libs-release-local:jmock/jmock/1.0.1/jmock-1.0.1.jar
Method Name: GET
User: resolver
Time: 2012-05-06T10:49:09.528+03:00
Thread: pool-1-thread-31
Steps:
2012-05-06T10:49:09.587+03:00 Received request
2012-05-06T10:49:09.588+03:00 Request source = 0:0:0:0:0:0:1, Last modified = 01-01-70 01:59:59 IST, If modified since = -1, Thread name = pool-1-thread-31
2012-05-06T10:49:09.697+03:00 Retrieving info
2012-05-06T10:49:09.723+03:00 Identified resource as a file
...
2012-05-06T10:49:09.788+03:00 Responding with selected content handle
2012-05-06T10:49:09.807+03:00 Request succeeded
```

Archive Entry Download

Description: Retrieves an archived resource from the specified archive destination.

Security: Requires a user with 'read' permission (can be anonymous)

Usage: GET /repo-key/path/to/artifact.jar!*/path/to/archived/resource (**NOTE!** the '!' between the archive file name and the archive entry path)

Sample Output:

```
GET
http://localhost:8081/artifactory/repol-cache/commons-lang/commons-lang/2.6/commons-lang-2.6.jar!/META-INF/LICENSE.txt
```

Create Directory

Description: Create new directory at the specified destination.

Notes: You can also [attach properties](#) as part of creating directories.

Security: Requires a user with 'deploy' permissions (can be anonymous)

Usage: PUT /repo-key/path/to/directory/

Produces: application/vnd.org.jfrog.artifactory.storage.ItemCreated+json

Sample Usage:

```
PUT /libs-release-local/path/to/directory/
{
  "uri":
  "http://localhost:8081/artifactory/libs-release-local/path/to/directory",
  "repo": "libs-release-local",
  "path": "/path/to/directory",
  "created": ISO8601 (yyyy-MM-dd'T'HH:mm:ss.SSSZ),
  "createdBy": "userY",
  "children" : [ ]
}
```

Deploy Artifact

Description: Deploy an artifact to the specified destination.

Notes: You can also [attach properties](#) as part of deploying artifacts.

Security: Requires a user with 'deploy' permissions (can be anonymous)

Usage: PUT /repo-key/path/to/artifact.ext

Produces: application/vnd.org.jfrog.artifactory.storage.ItemCreated+json

Sample Usage:

```
PUT /libs-release-local/my/jar/1.0/jar-1.0.jar
{
  "uri":
  "http://localhost:8081/artifactory/libs-release-local/my/jar/1.0/jar-1.0.jar",
  "downloadUri":
  "http://localhost:8081/artifactory/libs-release-local/my/jar/1.0/jar-1.0.jar",
  "repo": "libs-release-local",
  "path": "/my/jar/1.0/jar-1.0.jar",
  "created": ISO8601 (yyyy-MM-dd'T'HH:mm:ss.SSSZ),
  "createdBy": "userY",
  "size": "1024", //bytes
  "mimeType": "application/java-archive",
  "checksums":
  {
    "md5" : string,
    "sha1" : string
  },
  "originalChecksums":{
    "md5" : string,
    "sha1" : string
  }
}
```

Deploy Artifact by Checksum

Description: Deploy an artifact to the specified destination by checking if the artifact content already exists in Artifactory.

If Artifactory already contains a user readable artifact with the same checksum the artifact content is copied over to the new location and return a response without requiring content transfer.

Otherwise, a 404 error is returned to indicate that content upload is expected in order to deploy the artifact.

Notes: You can also [attach properties](#) as part of deploying artifacts.

Security: Requires a user with 'deploy' permissions (can be anonymous)

Usage: PUT /repo-key/path/to/artifact.ext

Headers: X-Checksum-Deploy: true, X-Checksum-Sha1: sha1Value, X-Checksum-Sha256: sha256Value, X-Checksum: checksum value (type is resolved by length)

Produces: application/vnd.org.jfrog.artifactory.storage.ItemCreated+json

Since: 2.5.1

Sample Output:

```
PUT /libs-release-local/my/jar/1.0/jar-1.0.jar
{
  "uri":
  "http://localhost:8081/artifactory/libs-release-local/my/jar/1.0/jar-1.0.jar",
  "downloadUri":
  "http://localhost:8081/artifactory/libs-release-local/my/jar/1.0/jar-1.0.jar",
  "repo": "libs-release-local",
  "path": "/my/jar/1.0/jar-1.0.jar",
  "created": ISO8601 (yyyy-MM-dd'T'HH:mm:ss.SSSZ),
  "createdBy": "userY",
  "size": "1024", //bytes
  "mimeType": "application/java-archive",
  "checksums":
  {
    "md5" : string,
    "sha1" : string
  },
  "originalChecksums":{
    "md5" : string,
    "sha1" : string
  }
}
```

Deploy Artifacts from Archive

Description: Deploys an archive containing multiple artifacts and extracts it at the specified destination maintaining the archive's file structure. Deployment is performed in a single HTTP request and only the extracted content is deployed, not the archive file itself.

Supported archive types are: zip; tar; tar.gz; and tgz. **NOTE!** that deployment of compressed archives (unlike tar) may incur considerable CPU overhead.

Notes: Requires Artifactory Pro

Security: Requires a user with 'deploy' permissions (can be anonymous)

Usage: PUT path1/to/repo-key/ /path2/to/archive.zip

Headers: X-Explode-Archive: true - archive will be exploded upon deployment, X-Explode-Archive-Atomic: true - archive will be exploded in an atomic operation upon deployment

Produces: text/plain

Since: 2.6.3

Sample Usage:

```
PUT /libs-release-local/ /Users/user/Desktop/archive.zip
```

Push a Set of Artifacts to Bintray

Deprecated: This endpoint is deprecated and is replaced with [Distribute Artifact](#)

Description: Push a set of artifacts to Bintray as a version.

Uses a descriptor file (that must have 'bintray-info' in its filename and a .json extension) that was deployed to artifactory, the call accepts the full path to the descriptor as a parameter.

For more details, please refer to [Pushing a Set of Files](#).

Signing a version is controlled by the `gpgSign` parameter in the descriptor file, and the `gpgSign` parameter passed to this command. **The value passed to this command always takes precedence over the value in the descriptor file.**

If you also want a passphrase to be applied to your signature, specify `gpgPassphrase=<passphrase>`.

Security: Requires a valid user with deploy permissions and Bintray credentials defined (for more details, please refer to [Entering your Bintray credentials](#)).

Usage: `POST /api/bintray/push?descriptor=pathToDescriptorFile[&gpgPassphrase=passphrase][&gpgSign=true/false]`

Since: 3.5.0

Produces: `application/vnd.org.jfrog.artifactory.bintray.BintrayPushResponse+json`

Sample Output:

```
{"Message": "Pushing build to Bintray finished successfully."}
```

Push Docker Tag to Bintray

Description: Push Docker tag to Bintray

Calculation can be synchronous (the default) or asynchronous.

Security: Requires a valid user with deploy permissions and Bintray credentials defined (for more details, please refer to [Entering your Bintray credentials](#)).

Usage: `POST /api/bintray/docker/push/{repoKey}`

Since: 3.6.0

Produces: `text/plain`

Sample Output:

```
POST api/bintray/docker/push/docker-local
{
  "dockerImage": "jfrog/ubuntu:latest", // The docker image to push, use
  ':' for specific tag or leave blank for 'latest'
  "bintraySubject": "shayy", // The Bintray Subject
  "bintrayRepo": "containers", // The Bintray Subject's repository
  "async": false // Optionally execute the push asynchronously. Default:
false
}
```

Distribute Artifact

Description: Deploys artifacts from Artifactory to Bintray, and creates an entry in the corresponding Artifactory distribution repository specified

Notes: Requires Artifactory Pro

Since: 4.8

Security: Requires an authenticated user.

Usage: `POST /api/distribute`

Consumes: `application/json`

```

{
  "publish" : "<true | false>"      // Default: true. If true, artifacts are
published when deployed to Bintray
  "overrideExistingFiles" : "<true | false>" // Default: false. If true,
Artifactory overwrites files already existing in the target path in
Bintray.
      // Existing version attributes are also overridden if defined
in the distribution repository Advanced Configuration
  "gpgPassphrase" : "<passphrase>"    // If specified, Artifactory will GPG
sign the version deployed to Bintray and apply the specified passphrase
  "async" : "<true | false>"          // Default: false. If true, the artifact
will be distributed asynchronously. Errors and warnings may be viewed in
the log file
  "targetRepo" : "<targetDistributionRepo>", // The Distribution Repository
into which artifacts should be deployed
  "packagesRepoPaths" : ["<localRepo/path/to/distribute>",
"<distRepo/path/to/distribute>"] // An array of local or distribution
repositories and corresponding paths to artifacts that should be deployed
to the specified target repository in Bintray
  "dryRun" : "<true | false>"        // Default: false. If true, distribution
is only simulated. No files are actually moved.
}

```

Sample input:

```

POST /api/distribute
{
  "targetRepo" : "dist-repo-jfrog-artifactory",
  "packagesRepoPaths" : ["yum-local/jfrog-artifactory-pro-4.7.6.rpm"]
}

```

File Compliance Info

Description: Get compliance info for a given artifact path. The result includes license and vulnerabilities, if any. Supported by local and local-cached repositories.

Notes: Requires Artifactory Pro, requires Black Duck addon enabled.

Since: 3.0.0

Security: Requires an authenticated user.

Usage: GET: /api/compliance/{repoKey}/{item-path}

Produces: application/json

Sample output:

```
GET:
/api/compliance/libs-release-local/ch/qos/logback/logback-classic/0.9.9/logback-classic-0.9.9.jar
{
  "licenses" : [ {"name": "LGPL v3", "url": "http://"}, {"name": "APL v2", "url": "http://"}... ],
  "vulnerabilities" : [ {"name": "CVE-13427", "url": "http://"}, {"name": "CVE-1041", "url": "http://"}... ]
}
```

Delete Item

Description: Deletes a file or a folder from the specified destination.

Security: Requires a user with 'delete' permission (can be anonymous)

Usage: DELETE /repo-key/path/to/file-or-folder

Sample Usage:

```
DELETE
http://localhost:8081/artifactory/libs-release-local/ch/qos/logback/logback-classic/0.9.9
```

Copy Item

Description: Copy an artifact or a folder to the specified destination. Supported by local repositories only.

Optionally suppress cross-layout module path translation during copy.

You can test the copy using a dry run.

Copy item behaves similarly to a standard file system and supports renames. If the target path does not exist, the source item is copied and optionally renamed. Otherwise, if the target exists and it is a directory, the source is copied and placed under the target directory.

Notes: Requires Artifactory Pro

Security: Requires a privileged user (can be anonymous)

Usage: POST /api/copy/{srcRepoKey}/{srcFilePath}?to={targetRepoKey}/{targetFilePath}&dry=1]&suppressLayouts=0/1(default)]&failFast=0/1

Produces: application/vnd.org.jfrog.artifactory.storage.CopyOrMoveResult+json

Since: 2.2.2

Sample Output:

```
POST
/api/copy/libs-release-local/org/acme?to=/ext-releases-local/org/acme-new&dry=1
{
  "messages" : [
    {
      "level": "error",
      "message": "The repository has denied...."
    },...
  ]
}
```

Move Item

Description: Moves an artifact or a folder to the specified destination. Supported by local repositories only.

Optionally suppress cross-layout module path translation during move.

You can test the move using dry run.

Move item behaves similarly to a standard file system and supports renames. If the target path does not exist, the source item is moved and optionally renamed. Otherwise, if the target exists and it is a directory, the source is moved and placed under the target directory.

Notes: Requires Artifactory Pro

Security: Requires a privileged user (can be anonymous)

Usage: POST /api/move/{srcRepoKey}/{srcFilePath}?to=/{targetRepoKey}/{targetFilePath}[%amp;dry=1][%amp;suppressLayouts=0/1 (default)][%amp;failFast=0/1]

Produces: application/vnd.org.jfrog.artifactory.storage.CopyOrMoveResult+json

Since: 2.2.2

Sample Output:

```
POST
/api/move/libs-release-local/org/acme?to=/ext-releases-local/org/acme-new&
dry=1
{
  "messages" : [
    {
      "level": "error",
      "message": "The repository has denied...."
    }, ...
  ]
}
```

Get Repository Replication Configuration

Description: Returns the replication configuration for the given repository key, if found. Supported by local and remote repositories. Note: The 'enableEventReplication' parameter refers to both push and pull replication.

Notes: Requires Artifactory Pro

Security: Requires a privileged user

Usage: GET /api/replications/{repoKey}

Produces: application/vnd.org.jfrog.artifactory.replications.ReplicationConfigRequest+json

Since: 3.1.1

Sample Usage:

```
GET /api/replications/libs-release-local
{
  "url" : "http://localhost:8081/artifactory/remote-repo",
  "socketTimeoutMillis" : 15000,
  "username" : "admin",
  "password" : "password",
  "enableEventReplication" : false,
  "enabled" : true,
  "cronExp" : "0 0 12 * * ?",
  "syncDeletes" : true,
  "syncProperties" : true,
  "syncStatistics" : false,
  "repoKey" : "libs-release-local",
  "pathPrefix" : "/path/to/repo"
}
```

Set Repository Replication Configuration

Description: Add or replace replication configuration for given repository key. Supported by local and remote repositories. Accepts the JSON

payload returned from [Get Repository Replication Configuration](#) for a single and an array of configurations. If the payload is an array of replication configurations, then values for `cronExp` and `enableEventReplication` in the first element in the array will determine the corresponding values when setting the repository replication configuration.

Notes: Requires Artifactory Pro

Security: Requires a privileged user

Usage: PUT /api/replications/{repoKey}

Consumes: application/vnd.org.jfrog.artifactory.replications.ReplicationConfigRequest+json

Since: 3.1.1

Sample Usage:

```
PUT /api/replications/libs-release-local
```

Update Repository Replication Configuration

Description: Update existing replication configuration for given repository key, if found. Supported by local and remote repositories.

Notes: Requires Artifactory Pro

Security: Requires a privileged user

Usage: POST /api/replications/{repoKey}

Consumes: full or partial application/vnd.org.jfrog.artifactory.replications.ReplicationConfigRequest+json

Since: 3.1.1

Sample Usage:

```
POST /api/replications/libs-release-local
```

Delete Repository Replication Configuration

Description: Delete existing replication configuration for given repository key. Supported by local and local-cached repositories.

Notes: Requires Artifactory Pro

Security: Requires a privileged user

Usage: DELETE /api/replications/{repoKey}

Since: 3.1.1

Sample Usage:

```
DELETE /api/replications/libs-release-local
```

Scheduled Replication Status

Description: Returns the status of scheduled [cron-based](#) replication jobs define via the Artifactory UI on repositories. Supported by local, local-cached and remote repositories.

Notes: Requires Artifactory Pro

Security: Requires a user with 'read' permission (can be anonymous)

Usage: GET /api/replication/{repoKey}

Produces: application/vnd.org.jfrog.artifactory.replication.ReplicationStatus+json

```

GET /api/replication/remote-libs
{
  "status": {status},
  "lastCompleted": {time},
  "targets":
  [
    { "url" : targetUrl, "repoKey": {repoKy}, "status" : {status},
    "lastCompleted" : {time} },
    ...
    { "url" : targetUrl, "repoKey": {repoKy}, "status" : {status},
    "lastCompleted" : {time}}
  ],
  "repositories":
  {
    {repoKy} : { "status" : {status}, "lastCompleted" : {time} },
    ...
    {repoKy} : { "status" : {status}, "lastCompleted" : {time} }
  }
}

```

where:

{status}= never_run|incomplete(running or interrupted)|error|warn|ok|inconsistent

{time}= time in ISO8601 format (yyyy-MM-dd'T'HH:mm:ss.SSSZ), or null if never completed

Since: 2.4.2

Sample Usage:

```

GET /api/replication/remote-libs
{
  "status" : "ok",
  "lastCompleted" : 2015-12-27T15:08:49.050+02:00",
  "targets":
  [
    { "url": "http://remote_host/remote-libs1", "repoKey": "remote-libs1",
    "status" : {status}, "lastCompleted" : "2015-12-27T15:07:49.050+02:00" },
    ...
    { "url" : "http://remote_host/remote-libs2", "repoKey": "remote-libs2",
    "status" : {status}, "lastCompleted" : "2015-12-27T15:07:49.050+02:00" }
  ],
  "repositories":
  {
    "remote-libs1" : { "status" : "ok", "lastCompleted" :
    "2015-12-27T15:07:49.050+02:00" },
    ...
    "remote-libs2" : { "status" : "ok", "lastCompleted" :
    "2015-12-27T15:07:49.050+02:00" }
  }
}

```

Pull/Push Replication

Description: Schedules immediate content replication between two Artifactory instances.

Replication can optionally include properties and delete items if they do not exist in the source repository.

This API completes the existing [cron-based](#) replication exposed via the Artifactory UI and allows for pre-scheduled execution.

Pull Replication - pulls content from a remote Artifactory repository to a local cache of the remote repository.

Push Replication - pushes content from a local repository into a local repository of another Artifactory instance.

Multi-push Replication - pushes content from a local repository into a local repository of several Artifactory instances. This feature is only available with Artifactory Enterprise license.

The type of replication initiated depends on the type of repository specified in the `repoPath` parameter.

If `repoPath` is a local repository, a push replication will be triggered. You may specify multiple target repositories in the payload for multi-push replication, but all must be local to their respective instances.

If `repoPath` is a remote repository cache, a pull replication will be triggered. Note that in this case you may only specify a single repository in the payload.

Important note - If no repositories are provided in the payload, Artifactory will trigger all existing replication configurations.

Security: Requires a privileged user (can be anonymous) For non-admin users, the maximum number of files that will be replicated is as defined by the `artifactory.search.userQueryLimit` system property.

Usage: POST `/api/replication/execute/{repoPath}`

Consumes: application/json

```
[
  {
+   "url" : "<URL of the repository at the remote Artifactory instance, Used
only by push replication>",
+   "username" : "<username at the remote Artifactory instance, Used only by
push replication>",
+   "password" : "<password at the remote Artifactory instance, Used only by
push replication>",
-   "proxy" : "<name of the proxy (if used) at the remote Artifactory
instance, Used only by push replication>"
-   "properties" : "<true | false>", // When true, properties of replicated
artifacts will be synchronized also
-   "delete" : "<true | false>" // When true, items that were deleted
remotely will also be deleted locally (including properties metadata)
  }
]
```

`+ =mandatory; -=optional`

Since: 4.7.5

Sample Usage:

```
// Single push replication
POST /api/replication/execute/libs-release-local
{
  [
    {
      "url": "http://localhost:8082/artifactory/libs-release-local",
      "username": "admin",
      "password": "password",
      "proxy": "localProxy"
    }
  ]
}
```

```
// Pull replication
POST /api/replication/execute/libs-remote
{
  [
    {
      "properties" : "true",
      "delete" : "true"
    }
  ]
}

// Multi-push replication
POST /api/replication/execute/libs-release-local
{
  [
    {
      "url": "http://localhost:8082/artifactory/libs-release-local",
      "username": "admin",
      "password": "password",
      "proxy": "localProxy",
      "properties" : "true",
      "delete" : "true"
    },
    {
      "url": "http://localhost:8082/artifactory/ext-release-local",
      "username": "admin",
      "password": "password"
      "properties" : "true",
      "delete" : "true"
    },
    {
      "url": "http://localhost:8082/artifactory/plugins-release-local",
      "username": "admin",
      "password": "password"
      "properties" : "true",
      "delete" : "true"
    }
  ]
}

// Trigger configured push replication
POST /api/replication/execute/libs-release-local
```

```
// Trigger configured pull replication
POST /api/replication/execute/libs-remote
```

Pull/Push Replication (Deprecated)

Description: Schedules immediate content replication between two Artifactory instances. Replication can include properties and can optionally delete local items if they do not exist in the source repository.

This API completes the existing [cron-based](#) replication exposed via the Artifactory UI and allows for on-demand execution.

Pull Replication - pulls content from a remote Artifactory repository to a local cache of the remote repository.

Push Replication - pushes content from a local repository into a remote Artifactory local repository.

Supported by local, local-cached and remote repositories.

Notes: Requires Artifactory Pro

Security: Requires a privileged user (can be anonymous) For non-admin users will replicate at max the number of files as defined by the `artifactory.search.userQueryLimit` system property.

Usage: POST /api/replication/{srcRepoKey}/{srcPath}

Consumes: application/vnd.org.jfrog.artifactory.replication.ReplicationRequest+json

Since: 2.4.0

Sample Usage:

```
POST /api/replication/libs-release-local/com/acme
{
  //The following is only applicable for push replication
  + "url" : "https://repo.nmiy.org/repo-key", // The remote repository URL
  + "username": "replicator", //The name of a user with deploy permissions
  on the remote repository
  + "password": "****", //The remote repository password
  - "properties": true, //Sync item properties (true by default)
  - "delete": true, //Sync deletions (false by default)
  - "proxy": "org-prox", //A name of an Artifactory-configured proxy to use
  for remote requests
}
```

`+=mandatory; -=optional`

Create or Replace Local Multi-push Replication

Description: Creates or replaces a local multi-push replication configuration. Supported by local and local-cached repositories.

Notes: Requires an enterprise license

Security: Requires an admin user.

Usage: PUT /api/replications/multiple/{repo-key}

Consumes: application/vnd.org.jfrog.artifactory.replications.MultipleReplicationConfigRequest+json

Since: 3.7

Sample Usage:

```

PUT /api/replications/multiple/libs-release-local
{
  "cronExp":"0 0/9 14 * * ?",
  "enableEventReplication":true,
  "replications":[
    {
+   "url": "http://localhost:8081/artifactory/repo-k",
+   "socketTimeoutMillis": 15000,
+   "username": "admin",
+   "password": "password",
-   "enableEventReplication": true,
-   "enabled": true,
-   "syncDeletes": false,
-   "syncProperties": true,
-   "syncStatistics" : false,
-   "repoKey": "libs-release-local"
    }
  ,
    {
+   "url": "http://localhost:8081/artifactory/repo-v",
+   "socketTimeoutMillis": 15000,
+   "username": "admin",
+   "password": "password",
-   "enableEventReplication": true,
-   "enabled": true,
-   "syncDeletes": false,
-   "syncProperties": true,
-   "syncStatistics" : false,
-   "repoKey": "libs-release-local"
    }
  ]
}

```

+ =mandatory; -=optional

Update Local Multi-push Replication

Description: Updates a local multi-push replication configuration. Supported by local and local-cached repositories.

Notes: Requires an enterprise license

Security: Requires an admin user.

Usage: POST /api/replications/multiple/{repo-key}

Consumes: application/vnd.org.jfrog.artifactory.replications.MultipleReplicationConfigRequest+json

Since: 3.7

Sample Usage:

```

POST /api/replications/multiple/libs-release-local
{
  "cronExp":"0 0/9 14 * * ?",
  "enableEventReplication":true,
  "replications":[
    {
+   "url": "http://localhost:8081/artifactory/repo-k",
+   "socketTimeoutMillis": 15000,
+   "username": "admin",
+   "password": "password",
-   "enableEventReplication": true,
-   "enabled": true,
-   "syncDeletes": false,
-   "syncProperties": true,
-   "syncStatistics" : false,
-   "repoKey": "libs-release-local"
    }
  ,
    {
+   "url": "http://localhost:8081/artifactory/repo-v",
+   "socketTimeoutMillis": 15000,
+   "username": "admin",
+   "password": "password",
-   "enableEventReplication": true,
-   "enabled": true,
-   "syncDeletes": false,
-   "syncProperties": true,
-   "syncStatistics" : false,
-   "repoKey": "libs-release-local"
    }
  ]
}

```

+ =mandatory; -=optional

Delete Local Multi-push Replication

Description:Deletes a local multi-push replication configuration. Supported by local and local-cached repositories.

Notes: Requires an enterprise license

Security: Requires an admin user.

Usage: DELETE /api/replications/{repoKey}?url={replicatedURL}

If the url parameter is omitted, all multi-push replication configurations for the source repository are deleted.

Produces: application/vnd.org.jfrog.artifactory.replications.ReplicationConfigRequest+json, application/vnd.org.jfrog.artifactory.replications.MultipleReplicationConfigRequest+json

Since: 3.7

Sample Usage:


```
DELETE
/api/replications/libs-release-local?url=http://10.0.0.1/artifactory/libs-
release-local

//Delete all multi-push replication configurations for libs-release-local
DELETE /api/replications/libs-release-local
```

Enable or Disable Multiple Replications

Description: Enables/disables multiple replication tasks by repository or Artifactory server based in include and exclude patterns.

Notes: Requires Artifactory Pro

Security: Requires a privileged user

Usage: POST /api/replications/{enable | disable}

Consumes: application/json

Since: 4.4.3

Sample Usage:

```
//Enable/disable all push replications except those going out to
http://artimaster:port/artifactory and
https://somearti:port/artifactory/local-repo.
POST /api/replications/{enable | disable}
{
  "include" : [ "*" ],
  "exclude" : [ "http://artimaster:port/artifactory/**",
"https://somearti:port/artifactory/local-repo" ]
}

//Enable/disable all push replications expect those going out to
http://artidr:port/artifactory
POST /api/replications/{enable | disable}
{
  "include" : [ "*" ],
  "exclude" : [ "http://artidr:port/artifactory/**" ]
}
```

Get Global System Replication Configuration

Description: Returns the global system replication configuration status, i.e. if push and pull replications are blocked or unblocked.

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: GET /api/system/replications

Produces: application/json

Since: 4.7.2

Sample Usage:

```
GET /api/system/replications
{
  "blockPullReplications": false,
  "blockPushReplications": false
}
```

Block System Replication

Description: Blocks replications globally. Push and pull are true by default. If false, replication for the corresponding type is not blocked.

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: POST `api/system/replications/block?push=[true|false]&pull=[true|false]`

Produces: text/plain

Since: 4.7.2

Sample Usage:

```
POST /api/system/replications/block
Successfully blocked all replications, no replication will be triggered.
```

Unblock System Replication

Description: Unblocks replications globally. Push and pull are true by default. If false, replication for the corresponding type is not unblocked.

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: POST `api/system/replications/unblock?push=[true|false]&pull=[true|false]`

Produces: text/plain

Since: 4.7.2

Sample Usage:

```
POST /api/system/replications/unblock
Successfully unblocked all replications
```

Artifact Sync Download

Description: Downloads an artifact with or without returning the actual content to the client. When tracking the progress marks are printed (by default every 1024 bytes). This is extremely useful if you want to trigger downloads on a remote Artifactory server, for example to force eager cache population of large artifacts, but want to avoid the bandwidth consumption involved in transferring the artifacts to the triggering client. If no content parameter is specified the file content is downloaded to the client.

Notes: This API requires Artifactory Pro.

Security: Requires a privileged user (can be anonymous)

Usage: GET `api/download/{repoKey}/{filePath}[?content=none/progress][&mark=numOfBytesToPrintANewProgressMark]`

Produces: application/octet-stream, text/plain (depending on content type)

Since: 2.2.2

Sample Output:

```
GET
/api/download/my-remote/org/acme/1.0/acme-1.0.jar?content=progress&mark=512
.....
.....
.....

Completed: 150/340 bytes
```

Folder Sync (Deprecated)

Description: Triggers a no-content download of artifacts from a remote Artifactory repository for all artifacts under the specified remote folder. Can optionally delete local files if they do not exist in the remote folder,

overwrite local files only if they are older than remote files or never overwrite local files.

The default is not to delete any local files and to overwrite older local files with remote ones. By default progress marks of the sync are displayed. The default timeout for the remote file list is 15000 milliseconds (15 seconds).

Notes: This API is **deprecated**. Requires Artifactory Pro

Security: Requires a privileged user (can be anonymous) For non-admin users will replicate at max the number of files as defined by the `artifactory.search.userQueryLimit` system property.

Usage: GET

`/api/sync/{remoteRepositoryKey}/{folderPath}[?progress=showProgress][&mark=numOfBytesToPrintANewProgressMark][&delete=deleteExistingFiles][&overwrite=never/force][&timeout=fileListTimeoutInMillis]`

Produces: text/plain

Since: 2.2.4

Sample Output:

```
GET /api/sync/my-remote/org/acme/1.0?progress=1&delete=1
.....
.....
.....
.....

Completed: 970/1702 bytes
.....
.....

Completed: 1702/1702 bytes
Completed with 0 errors and 2 warnings (please check the server log for
more details).
```

File List

Description: Get a flat (the default) or deep listing of the files and folders (not included by default) within a folder.

For deep listing you can specify an optional depth to limit the results.

Optionally include a map of metadata timestamp values as part of the result (only properties are displayed in since 3.0.0).

folder inclusion since 2.3.2; checksum inclusion since: 2.3.3; include folder root path since: 2.5.2. Supported by all types of repositories.

Since: 2.2.4

Notes: Requires Artifactory Pro

Security: Requires a non-anonymous privileged user.

Usage: GET `/api/storage/{repoKey}/{folder-path}?list[&deep=0/1][&depth=n][&listFolders=0/1][&mdTimestamps=0/1][&includeRootPath=0/1]`

Produces: application/vnd.org.jfrog.artifactory.storage.FileList+json

Sample Output:

```

GET
/api/storage/libs-release-local/org/acme?list&deep=1&listFolders=1&mdTimes
tamps=1
{
  "uri":
"http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme
",
  "created": ISO8601,
  "files" : [
    {
      "uri": "/archived",
      "size": "-1",
      "lastModified": ISO8601,
      "folder": "true"
    },
    {
      "uri": "/doc.txt",
      "size": "253207", //bytes
      "lastModified": ISO8601,
      "folder": "false",
      "sha1": sha1Checksum,
      "mdTimestamps": { "properties" : lastModified (ISO8601) }
    },
    {
      "uri": "/archived/doc1.txt",
      "size": "253100", //bytes
      "lastModified": ISO8601,
      "folder": "false",
      "sha1": sha1Checksum,
      "mdTimestamps": { "properties" : lastModified (ISO8601) }
    },...
  ]
}

```

Get Background Tasks

Description: Retrieves list of background tasks currently scheduled or running in Artifactory. In HA, the nodeId is added to each task. Task can be in one of few states: scheduled, running, stopped, cancelled. Running task also shows the task start time.

Since: 4.4.0

Security: Requires a valid admin user

Usage: GET /api/tasks

Produces: application/json

Sample Output:

```
{
  "tasks" : [ {
    "id" :
    "artifactory.UpdateIndicesJob#d7321feb-6fd9-4e27-8f0e-954137be855b",
    "type" :
    "org.artifactory.addon.gems.index.GemsVirtualIndexHandler$updateIndicesJob",
    "state" : "scheduled",
    "description" : "Gems Virtual Repositories Index Calculator",
    "nodeId" : "artifactory-primary"
  }, {
    "id" :
    "artifactory.VirtualCacheCleanupJob#82bb1514-ea34-4a71-940d-78a61887981e",
    "type" : "org.artifactory.repo.cleanup.VirtualCacheCleanupJob",
    "state" : "scheduled",
    "description" : "",
    "nodeId" : "artifactory-primary"
  }, {
    "id" :
    "artifactory.BinaryStoreGarbageCollectorJob#039664ac-990d-4a32-85e1-decd0b508142",
    "type" :
    "org.artifactory.storage.binstore.service.BinaryStoreGarbageCollectorJob",
    "state" : "running",
    "started" : "2015-05-15T15:39:37.566+02:00",
    "description" : "Binaries Garbage Collector",
    "nodeId" : "artifactory-primary"
  } ]
}
```

Empty Trash Can

Description: Empties the trash can permanently deleting all its current contents.

Notes: Requires Artifactory Pro

Security: Requires a valid admin user

Usage: POST /api/trash/empty

Since: 4.4.3

Delete Item From Trash Can

Description: Permanently deletes an item from the trash can.

Notes: Requires Artifactory Pro

Security: Requires a valid admin user

Usage: DELETE /api/trash/clean/{repoName/path}

Since: 4.4.3

Sample usage:

```
DELETE /api/trash/clean/npm-local
```

Restore Item from Trash Can

Description: Restore an item from the trash can.

Notes: Requires Artifactory Pro

Security: Requires a valid admin user

Usage: POST /api/trash/restore/{from path}?to={to path}

Since: 4.4.3

Sample usage:

```
POST /api/trash/restore/npm-local?to=npm-local2
```

```
Successfully restored trash items
```

Optimize System Storage

Description: Raises a flag to invoke balancing between redundant storage units of a sharded filestore following the next garbage collection.

Since: 4.6.0

Notes: This is an advanced feature intended for administrators.

Security: Requires a valid admin user.

Usage: POST /api/system/storage/optimize

Produces: text/plain

Sample Usage:

```
POST /api/system/storage/optimize
```

```
200 OK
```

Get Puppet Modules

Description: Returns a list of all Puppet modules hosted by the specified repository. Results are paginated and all of the parameters in the pagination section are optional.

Notes: Requires Artifactory Pro. This endpoint will work only on local and remote repositories.

Usage: GET /api/puppet/{repoKey}/v3/modules

Security: Requires a privileged user (can be anonymous)

Produces: application/json

↕ [Click here to expand...](#)

```
{
  "total": 0,
  "limit": 0,
  "offset": 0,
  "current": "uri",
  "next": "uri",
  "previous": "uri",
  "results": [
    {
      "uri": "uri",
      "name": "",
      "downloads": 0,
      "created_at": "date-time",
      "updated_at": "date-time",
      "supported": false,
      "owner": {
        "uri": "",
        "username": ""
      },
      "current_release": {
        "uri": "",
        "version": "",
        "module": "object",
        "metadata": "object",
        "tags": [
          ""
        ],
        "supported": false,
        "file_size": 0,
        "file_md5": "",
        "downloads": 0,
        "readme": "",
        "changelog": "",
        "license": "",
        "created_at": "date-time",
        "updated_at": "date-time",
        "deleted_at": "date-time"
      },
      "releases": [
        {
          "uri": "uri",
          "version": ""
        }
      ],
      "homepage_url": "uri",
      "issues_url": "uri"
    }
  ]
}
```

Sample Usage:

```
GET /api/puppet/puppet-local/v3/modules/
Response:
{
  "pagination" : {
    "limit" : 20,
    "offset" : 0,
    "first" : "/v3/modules?limit=20&offset=0",
    "previous" : null,
    "current" : "/v3/modules?limit=20&offset=0",
    "next" : null,
    "total" : 1
  },
  "results" : [ {
    "uri" : "/v3/modules/maestrodev-wget",
    "slug" : "maestrodev-wget",
    "name" : "wget",
    "downloads" : 0,
    "created_at" : "2017-07-16 12:07:715 +0300",
    "updated_at" : "2017-07-16 12:07:00 +0300",
    "supported" : false,
    "endorsement" : null,
    "module_group" : "base",
    "owner" : {
      "uri" : "/v3/users/maestrodev",
      "slug" : "maestrodev",
      "username" : "maestrodev",
      "gravatar_id" : null
    },
    "current_release" : {
      "uri"
    }
  } ]
}
```

Get Puppet Module

Description: Returns information about a specific Puppet module.

Notes: Requires Artifactory Pro. This endpoint will work only on local and remote repositories.

Usage: GET /api/puppet/{repoKey}/v3/modules/{user}-{module}

Security: Requires a privileged user (can be anonymous)

Produces: application/json

▼ [Click here to expand...](#)


```

{
  "uri": "uri",
  "name": "",
  "downloads": 0,
  "created_at": "date-time",
  "updated_at": "date-time",
  "supported": false,
  "owner": {
    "uri": "",
    "username": ""
  },
  "current_release": {
    "uri": "",
    "version": "",
    "module": "object",
    "metadata": "object",
    "tags": [
      ""
    ],
    "supported": false,
    "file_size": 0,
    "file_md5": "",
    "downloads": 0,
    "readme": "",
    "changelog": "",
    "license": "",
    "created_at": "date-time",
    "updated_at": "date-time",
    "deleted_at": "date-time"
  },
  "releases": [
    {
      "uri": "uri",
      "version": ""
    }
  ],
  "homepage_url": "uri",
  "issues_url": "uri"
}

```

Get Puppet Releases

Description: Returns a list of all Puppet releases hosted by the specified repository. Results are paginated and all of the parameters in the pagination section are optional.

Notes: Requires Artifactory Pro. This endpoint will work only on local and remote repositories.

Usage: GET /api/puppet/{repoKey}/v3/releases

Security: Requires a privileged user (can be anonymous)

Produces: application/json

```
{
  "total": 0,
  "limit": 0,
  "offset": 0,
  "current": "uri",
  "next": "uri",
  "previous": "uri",
  "results": [
    {
      "uri": "uri",
      "version": "",
      "module": {
        "uri": "",
        "name": ""
      },
      "metadata": "object",
      "tags": [
        ""
      ],
      "supported": false,
      "file_size": 0,
      "file_md5": "",
      "downloads": 0,
      "readme": "",
      "changelog": "",
      "license": "",
      "created_at": "date-time",
      "updated_at": "date-time",
      "deleted_at": "date-time"
    }
  ]
}
```

Get Puppet Release

Description: Returns information about the specific Puppet module's release.

Notes: Requires Artifactory Pro. This endpoint will work only on local and remote repositories.

Usage: GET /api/puppet/{repoKey}/v3/releases/{user}-{module}-{version}

Security: Requires a privileged user (can be anonymous)

Produces: application/json

```

{
  "uri": "uri",
  "version": "",
  "module": {
    "uri": "",
    "name": ""
  },
  "metadata": "object",
  "tags": [
    ""
  ],
  "supported": false,
  "file_size": 0,
  "file_md5": "",
  "downloads": 0,
  "readme": "",
  "changelog": "",
  "license": "",
  "created_at": "date-time",
  "updated_at": "date-time",
  "deleted_at": "date-time"
}

```

SEARCHES

All searches return limited results for internal and anonymous users (same limits as in the user interface).

To modify the default limit results, edit the `artifactory.system.properties` file with `artifactory.search.limitAnonymousUsersOnly=false` (default is `true`) and add a new limit with `artifactory.search.userQueryLimit` (default is 1000).

Applicable to the following REST API calls:

Artifact Search, Archive Entries Search, GAVC Search, Property Search, Checksum Search (limited by UI max results), Artifacts Not Downloaded Since, Artifacts With Date in Date Range, Artifacts Created in Date Range.

Artifactory Query Language (AQL)

Description: Flexible and high performance search using [Artifactory Query Language \(AQL\)](#).

Since: 3.5.0

Security: Requires an authenticated user. Certain domains/queries may require Admin access.

Usage: POST `/api/search/aql`

Consumes: text/plain

Sample Usage:

```

POST /api/search/aql
items.find(
  {
    "repo": {"$eq": "libs-release-local"}
  }
)

```

Produces: application/json

Sample Output:

```

{
  "results" : [
    {
      "repo" : "libs-release-local",
      "path" : "org/jfrog/artifactory",
      "name" : "artifactory.war",
      "type" : "item type",
      "size" : "75500000",
      "created" : "2015-01-01T10:10:10",
      "created_by" : "Jfrog",
      "modified" : "2015-01-01T10:10:10",
      "modified_by" : "Jfrog",
      "updated" : "2015-01-01T10:10:10"
    }
  ],
  "range" : {
    "start_pos" : 0,
    "end_pos" : 1,
    "total" : 1
  }
}

```

Artifact Search (Quick Search)

Description: Artifact search by part of file name.

Searches return file info URIs. Can limit search to specific repositories (local or caches).

Since: 2.2.0

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/search/artifact?name=name[&repos=x[,y]]

Headers (Optionally): X-Result-Detail: info (To add all extra information of the found artifact), X-Result-Detail: properties (to get the properties of the found artifact), X-Result-Detail: info, properties (for both).

Produces: application/vnd.org.jfrog.artifactory.search.ArtifactSearchResult+json

Sample Output:

```

GET /api/search/artifact?name=lib&repos=libs-release-local
{
  "results": [
    {
      "uri":
      "http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme
      /lib/ver/lib-ver.pom"
    }, {
      "uri":
      "http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme
      /lib/ver2/lib-ver2.pom"
    }
  ]
}

```

Archive Entries Search (Class Search)

Description: Search archive for classes or any other resources within an archive.

Can limit search to specific repositories (local or caches).

Since: 2.2.0

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/search/archive?name=[archiveEntryName][&repos=x[,y]]

Produces: application/vnd.org.jfrog.artifactory.search.ArchiveEntrySearchResult+json

Sample Output:

```
GET
/api/search/archive?name=*Logger.class&repos=third-party-releases-local,repol-cache
{
  "results" : [
    {
      "entry":
      "org/apache/jackrabbit/core/query/lucene/AbstractIndex.LoggingPrintStream.class",
      "archiveUris": [
        "http://localhost:8081/artifactory/api/storage/third-party-releases-local/org/apache/jackrabbit/jackrabbit-core/1.2.3/jackrabbit-core-1.2.3.jar",
        "http://localhost:8081/artifactory/api/storage/third-party-releases-local/org/apache/jackrabbit/jackrabbit-core/1.3.1/jackrabbit-core-1.3.1.jar"
      ]
    }, {
      "entry": "org/codehaus/plexus/logging/AbstractLogger.class",
      "archiveUris": [
        "http://localhost:8081/artifactory/api/storage/repol-cache/org/codehaus/plexus/plexus-container-default/1.0-alpha-9-stable-1/plexus-container-default-1.0-alpha-9-stable-1.jar"
      ]
    }
  ]
}
```

GAVC Search

Description: Search by Maven coordinates: GroupId, ArtifactId, Version & Classifier.

Search must contain at least one argument. Can limit search to specific repositories (local and remote-cache).

Since: 2.2.0

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/search/gavc?[g=groupId][&a=artifactId][&v=version][&c=classifier][&repos=x[,y]]

Headers (Optionally): X-Result-Detail: info (To add all extra information of the found artifact), X-Result-Detail: properties (to get the properties of the found artifact), X-Result-Detail: info, properties (for both).

Produces: application/vnd.org.jfrog.artifactory.search.GavcSearchResult+json

Sample Output:

```

GET
/api/search/gavc?g=org.acme&a=artifact&v=1.0&c=sources&repos=libs-release-
local
{
  "results": [
    {
      "uri":
      "http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme
/artifact/1.0/artifact-1.0-sources.jar"
    }, {
      "uri":
      "http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme
/artifactB/1.0/artifactB-1.0-sources.jar"
    }
  ]
}

```

Property Search

Description: Search by properties.

If no value is specified for a property - assume "*". Can limit search to specific repositories (local, remote-cache or virtual).

Since: 2.2.0

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/search/prop?[p1=v1,v2][&p2=v3][&repos=x[,y]]

Headers (Optionally): X-Result-Detail: info (To add all extra information of the found artifact), X-Result-Detail: properties (to get the properties of the found artifact), X-Result-Detail: info, properties (for both).

Produces: application/vnd.org.jfrog.artifactory.search.MetadataSearchResult+json

Sample Output:

```

GET /api/search/prop?p1=v1,v2&p2=v3&repos=libs-release-local
{
  "results" : [
    {
      "uri":
      "http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme
/lib/ver/lib-ver.pom"
    }, {
      "uri":
      "http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme
/lib/ver2/lib-ver2.pom"
    }
  ]
}

```

Checksum Search

Description: Artifact search by checksum (md5, sha1, or sha256)

Searches return file info URIs. Can limit search to specific repositories (local, remote-cache or virtual).

Notes: Requires Artifactory Pro

Since: 2.3.0

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/search/checksum?md5=md5sum?sha1=sha1sum?sha256=sha256sum[&repos=x[,y]]

Headers (Optionally): X-Result-Detail: info (To add all extra information of the found artifact), X-Result-Detail: properties (to get the properties of

the found artifact), X-Result-Detail: info, properties (for both).

Produces: application/vnd.org.jfrog.artifactory.search.ChecksumSearchResult+json

Sample Output:

```
GET
/api/search/checksum?sha256=9a7fb65f15e00aa2a22c1917d0dafd4374fee8daf0966a
4d94cd37a0b9acafb9&repos=libs-release-local
{
  "results": [
    {
      "uri":
      "http://localhost:8081/artifactory/api/storage/libs-release-local/org/jfro
g/build-info-api/1.3.1/build-info-api-1.3.1.jar"
    }
  ]
}
```

Bad Checksum Search

Description: Find all artifacts that have a bad or missing client checksum values (md5 or sha1)
Searches return file info uris. Can limit search to specific repositories (local, remote-cache or virtual).

Notes: Requires Artifactory Pro

Since: 2.3.4

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/search/badChecksum?type=md5|sha1[&repos=x[,y]]

Produces: application/vnd.org.jfrog.artifactory.search.BadChecksumSearchResult+json

Sample Output:

```
GET /api/search/badChecksum?type=md5&repos=libs-release-local
{
  "results": [
    {
      "uri":
      "http://localhost:8081/artifactory/api/storage/libs-release-local/org/jfro
g/build-info-api/1.3.1/build-info-api-1.3.1.jar"
      "serverMd5": "4040c7c184620af0a0a8a3682a75eb7"
      "clientMd5": "4040c7c184620af0a0a8a3682a75e44" //On missing
checksum this element will be an empty string
    }
  ]
}
```

Artifacts Not Downloaded Since

Description: Retrieve all artifacts not downloaded since the specified Java epoch in **milliseconds**.

Optionally include only artifacts created before the specified `createdBefore` date, otherwise only artifacts created before `notUsedSince` are returned.

Can limit search to specific repositories (local or caches).

Since: 2.2.4

Security: Requires a privileged non-anonymous user.

Usage: GET /api/search/usage?notUsedSince=javaEpochMillis[&createdBefore=javaEpochMillis][&repos=x[,y]]

Produces: application/vnd.org.jfrog.artifactory.search.ArtifactUsageResult+json

Sample Output:

```
GET
/api/search/usage?notUsedSince=long&createdBefore=long&repos=libs-release-
local
{
  "results" : [
    {
      "uri":
      "http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme
      /lib/ver/lib-ver.jar",
      "lastDownloaded": ISO8601
    }, {
      "uri":
      "http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme
      /lib/ver2/lib-ver2.jar",
      lastDownloaded: ISO8601
    }
  ]
}
```

Artifacts With Date in Date Range

Description: Get all artifacts with specified dates within the given range. Search can be limited to specific repositories (local or caches).

Since: 3.2.1

Security: Requires a privileged non-anonymous user.

Usage: GET /api/search/dates?[from=fromVal][&to=toVal][&repos=x[,y]][&dateFields=c[,d]]

Parameters: The `from` and `to` parameters can be either a long value for the java epoch (**milliseconds** since the epoch), or an ISO8601 string value. `from` is mandatory. If `to` is not provided, `now()` will be used instead, and if either are omitted, `400 bad request` is returned.

The `dateFields` parameter is a comma separated list of date fields that specify which fields the `from` and `to` values should be applied to. The date fields supported are: `created`, `lastModified`, `lastDownloaded`.

It is a mandatory field and it also dictates which fields will be added to the JSON returned.

If ANY of the specified date fields of an artifact is within the specified range, the artifact will be returned.

Produces: application/vnd.org.jfrog.artifactory.search.ArtifactResult+json

Sample Output:


```
GET
/api/search/dates?dateFields=created,lastModified,lastDownloaded&from=long
&to=long&repos=libs-release-local
{
  "results" : [
    {
      "uri":
"http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme
/lib/ver/lib-ver.jar",
      "created": ISO8601,
      "lastModified": ISO8601,
      "lastDownloaded": ISO8601
    },{
      "uri":
"http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme
/lib/ver2/lib-ver2.jar",
      "created": ISO8601.
      "lastModified": ISO8601,
      "lastDownloaded": ISO8601
    }
  ]
}
```

Artifacts Created in Date Range

Description: Get All Artifacts Created in Date Range

If 'to' is not specified use now(). Can limit search to specific repositories (local or remote-cache).

Since: 2.2.0

Security: Requires a privileged non-anonymous user.

Usage: GET /api/search/creation?from=javaEpochMillis[&to=javaEpochMillis[&repos=x[,y]]

Produces: application/vnd.org.jfrog.artifactory.search.ArtifactCreationResult+json

Sample Output:

```
GET /api/search/creation?from=long&to=long&repos=libs-release-local
{
  "results" : [
    {
      "uri":
"http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme
/lib/ver/lib-ver.jar",
      "created": ISO8601
    },{
      "uri":
"http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme
/lib/ver2/lib-ver2.jar",
      "created": ISO8601
    }
  ]
}
```

Pattern Search

Description: Get all artifacts matching the given Ant path pattern

Since: 2.2.4

Notes: Requires Artifactory Pro. Pattern "***" is not supported to avoid overloading search results.

Security: Requires a privileged non-anonymous user.

Usage: GET /api/search/pattern?pattern=repo-key:this/is/a/*pattern*.war

Produces: application/vnd.org.jfrog.artifactory.search.PatternResultFileSet+json

Sample Output:

```
GET /api/search/pattern?pattern=libs-release-local:killer/*/ninja/*/*.jar
{
  "repositoryUri" :
  "http://localhost:8081/artifactory/libs-release-local",
  "sourcePattern" : "libs-release-local:killer/*/ninja/*/*.jar",
  files : [
    "killer/coding/ninja/1.0/monkey-1.0.jar",
    "killer/salty/ninja/1.5-SNAPSHOT/pickle-1.5-SNAPSHOT.jar"
  ]
}
```

Builds for Dependency

Description: Find all the builds an artifact is a dependency of (where the artifact is included in the build-info dependencies)

Notes: Requires Artifactory Pro

Since: 2.3.4

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/search/dependency?sha1=sha1Checksum

Produces: application/vnd.org.jfrog.artifactory.search.DependencyBuilds+json

Sample Output:

```
GET /api/search/dependency?sha1=451a3c5f8cfa44c5d805379e760b5c512c7d250b
{
  "results" : [
    {
      "uri": "http://localhost:8081/artifactory/api/build/my-build/50"
    }, {
      "uri": "http://localhost:8081/artifactory/api/build/my-build/51"
    }
  ]
}
```

License Search

Description: Search for artifacts that were already tagged with license information and their respective licenses.

To search by specific license values use Property Search with the 'artifactory.licenses' property.

When the autofind parameter is specified Artifactory will try to automatically find new license information and return it as part of the result in the found field.

Please note that this can affect the speed of the search quite dramatically, and will still search only on already-tagged artifacts.

Default parameter values when unspecified: unapproved=1, unknown=1, notfound=0, neutral=0, approved=0, autofind=0.

Can limit search to specific repositories (local, remote-cache or virtual).

Since: 2.3.0

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: GET /api/search/license?unapproved=1][&unknown=1][¬found=0][&neutral=0][&approved=0][&autofind=0][&repos=x[,y]]

Produces: application/vnd.org.jfrog.artifactory.search.LicenseResult+json

Sample Output:

```
GET
/api/search/license?approved=1&unknown=1&autofind=1&repos=libs-release-local, staging
{
  "results" : [
    {
      "uri":
      "http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme/lib/ver/lib-ver.jar",
      "license": "lgplv2",
      "found": "lgplv2",
      "status": "approved"
    }, {
      "uri":
      "http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme/lib/ver/lib-ver.jar",
      "license": "cddl1",
      "found": "gplv3",
      "status": "neutral"
    }, {
      "uri":
      "http://localhost:8081/artifactory/api/storage/staging/org/acme/lib/ver2/lib-ver2.jar",
      "license": "gplv3",
      "found": "gplv3",
      "status": "unapproved"
    }
  ]
}
```

Artifact Version Search

Description: Search for all available artifact versions by GroupId and ArtifactId in local, remote or virtual repositories.

Search can be limited to specific repositories (local, remote and virtual) by settings the `repos` parameter.

Release/integration versions: Unless the `version` parameter is specified, both release and integration versions are returned. When `version` is specified, e.g. `1.0-SNAPSHOT`, result includes only integration versions.

Integration versions are determined by the [repository layout](#) of the repositories searched. For integration search to work the repository layout requires an 'Artifact Path Pattern' that contains the `baseRev` token and then the `fileIntegRev` token with only literals between them.

Remote searches: By default only local and cache repositories are used. When specifying `remote=1`, Artifactory searches for versions on remote repositories. **NOTE!** that this can dramatically slow down the search.

For Maven repositories the remote `maven-metadata.xml` is consulted. For non-maven layouts, remote file listing runs for all remote repositories that have the 'List Remote Folder Items' checkbox enabled.

Filtering results (Artifactory 3.0.2+): The `version` parameter can accept the `*` and/or `?` wildcards which will then filter the final result to match only those who match the given version pattern.

Since: 2.6.0

Notes: Requires Artifactory Pro

Security: Requires a privileged user (can be anonymous)

Usage: `GET /api/search/versions?[g=groupId][&a=artifactId][&v=version][&remote=0/1][&repos=x[,y]]`

Produces: application/vnd.org.jfrog.artifactory.search.ArtifactVersionsResult+json

Sample Output:

```

GET /api/search/versions?g=org.acme&a=artifact&repos=libs-release-local
{
  "results": [
    {
      "version": "1.2",
      "integration": false
    }, {
      "version": "1.0-SNAPSHOT",
      "integration": true
    }, {
      "version": "1.0",
      "integration": false
    }
  ]
}

```

Artifact Latest Version Search Based on Layout

Description: Search for the latest artifact version by groupId and artifactId, based on the layout defined in the repository. Search can be limited to specific repositories (local, remote-cache or virtual) by settings the `repos` parameter. When searching in a virtual repository, each child-repository layout will be consulted accordingly.

Latest release vs. latest integration: Unless the `version` parameter is specified, the search returns the latest artifact release version. When `version` is specified, e.g. `1.0-SNAPSHOT`, the result is the latest integration version. Integration versions are determined by the [repository layout](#) of the repositories searched. For integration search to work the repository layout requires an "Artifact Path Pattern" that contains the `baseRev` token and then the `fileIntegRev` token with only literals between them.

Remote searches: By default only local and cache repositories will be used. When specifying `remote=1`, Artifactory searches for versions on remote repositories. **NOTE!** that this can dramatically slow down the search.

For Maven repositories the remote `maven-metadata.xml` will be consulted. For non-Maven layouts, remote file listing runs for all remote repositories that have the 'List Remote Folder Items' checkbox enabled.

Filtering results (Artifactory 3.0.2+): The `version` parameter can accept the `*` and/or `?` wildcards which will then filter the final result to match only those who match the given version pattern.

Artifact path pattern: The `[org]` and `[module]` fields must be specified in the `artifact path pattern` of the repository layout for this call to work.

Since: 2.6.0

Notes: Requires Artifactory Pro

Security: Requires a privileged user (can be anonymous)

Usage: `GET /api/search/latestVersion?g=groupId[&a=artifactId][&v=version][&remote=1][&repos=x[,y]]`

Produces: text/plain

Sample Output:

```

GET
/api/search/latestVersion?g=org.acme&a=artifact&v=1.0-SNAPSHOT&repos=libs-
snapshot-local

1.0-201203131455-2

```

Artifact Latest Version Search Based on Properties

Description: Search for artifacts with the latest value in the "version" property. **Only artifacts with a "version" property expressly defined in lower case will be returned.** Results can be filtered by specifying additional properties.

{repo}: Specify a repository to search through or replace with `"_any"` to search through all repositories

{path}: Specify a path to search through or replace with `"_any"` to search through all paths

listFiles=0 (default): Artifactory will only retrieve the latest version

listFiles=1: Artifactory will retrieve the latest version and the corresponding files

You may specify filters to restrict the set of artifacts that are searched by adding any properties to your search URL

Notes: Requires Artifactory Pro

Since: 3.1.1

Security: Requires an authenticated user (not anonymous) to use the api and read permission to the repository of each artifact.

Usage: GET /api/versions/{repo}/{path}?[listFiles=0/1]&[<property key>=<property value>]&[<property key>=<property value>]

Consumes: json

Examples:

```
Return the latest version and corresponding artifacts by searching for
through all repositories whose path starts with a/b and are annotated with
the properties os=win and license=GPL.
```

```
GET /api/versions/_any/a/b?os=win&license=GPL&listFiles=1
```

```
{
  "version" : "1.1.2",
  "artifacts" : [ {
    "uri" : "http://...."
  } ]
}
```

```
Return the latest version (without the corresponding artifacts) by
searching through all repositories whose path starts with a/b and are
annotated with the properties os=win and license=GPL.
```

```
Return only the version.
```

```
GET /api/versions/_any/a/b?os=win&license=GPL
```

```
{
  "version" : "1.1.2",
  "artifacts" : []
}
```

Build Artifacts Search

Description: Find all the artifacts related to a specific build.

Notes: Requires Artifactory Pro

Since: 2.6.5

Security: Requires a privileged user (can be anonymous)

Usage: POST /api/search/buildArtifacts

Consumes: application/vnd.org.jfrog.artifactory.search.BuildArtifactsRequest+json

Sample Usage:

```

POST /api/search/buildArtifacts
{
  +"buildName": "build-name" // The build name for search by
  +"buildNumber": "15" // The build number to search by, can be LATEST to
search for the latest build number
  -"buildStatus": "Released" // Optionally search by latest build status
(e.g: "Released")
  -"repos": ["libs-release-local,ext-release-local"] // Optionally refine
search for specific repos, omit to search within all repositories
  -"mappings": [ // Optionally refine the search by providing a list of
regex patterns to search by
    {
      "input": "(.+)-sources.jar"
    },
    {
      "input": "(.+)-javadoc.jar"
    }
  ]
}

```

Produces: application/vnd.org.jfrog.artifactory.search.BuildArtifactsSearchResult+json

Sample Output:

```

POST /api/search/buildArtifacts
{
  "results" : [
    {
      "downloadUri":
      "http://localhost:8081/artifactory/libs-release-local/org/acme/lib/ver/lib-
sources.jar"
    }, {
      "downloadUri":
      "http://localhost:8081/artifactory/ext-release-local/org/acme/lib/ver/lib-
ver-javadoc.jar"
    }
  ]
}

```

List Docker Repositories

Description: Lists all Docker repositories (the registry's `_catalog`) hosted in an Artifactory Docker repository.

Since: 4.4.3. The `n` and `last` pagination parameters are supported from version 5.4.6.

Notes: Requires Artifactory Pro

Security: Requires a privileged user

Usage: GET `/api/docker/{repo-key}/v2/_catalog?n=<n from the request>&last=<last tag value from previous response>`

Produces: application/json

```
{
  "repositories": [
    <name>,
    ...
  ]
}
```

Sample Usage:

```
GET /api/docker/docker-local/v2/_catalog
{
  "repositories": [
    "busybox",
    "centos",
    "hello-world"
  ]
}
```

List Docker Tags

Description: Lists all tags of the specified Artifactory Docker repository.

Since: 4.4.3. The `n` and `last` pagination parameters are supported from version 5.4.6.

Notes: Requires Artifactory Pro

Security: Requires a privileged user

Usage: GET /api/docker/{repo-key}/v2/{image name}/tags/list?n=<n from the request>&last=<last tag value from previous response>

Produces: application/json

```
{
  "name": "<image name>",
  "tags" : [ "<tag>" ]
}
```

Sample Usage:

```
GET api/docker/v2/postgres/tags/list
{
  "name" : "postgres",
  "tags" : [ "9.5.2" ]
}
```

SECURITY

Get Users

Description: Get the users list

Since: 2.4.0

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: GET /api/security/users

Produces: application/vnd.org.jfrog.artifactory.security.Users+json

Sample Output:

```
GET /api/security/users
[
  {
    "name": "davids"
    "uri" : "http://localhost:8081/artifactory/api/security/users/davids"
    "realm" : "internal"
  }, {
    "name": "danl"
    "uri" : "http://localhost:8081/artifactory/api/security/users/danl"
    "realm" : "ldap"
  }
]
```

Get User Details

Description: Get the details of an Artifactory user

Since: 2.4.0

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: GET /api/security/users/{userName}

Produces: application/vnd.org.jfrog.artifactory.security.User+json

Sample Output:

```
GET /api/security/users/davids
{
  user.json
}
```

Get User Encrypted Password

Description: Get the encrypted password of the authenticated requestor

Since: 3.3.0

Security: Requires a privileged user

Usage: GET /api/security/encryptedPassword

Produces: plain/text

Sample Output:

```
GET /api/security/encryptedPassword

AP5v2zs9ga7CJNZb74u3arAKE5B
```

Create or Replace User

Description: Creates a new user in Artifactory or replaces an existing user

Since: 2.4.0

Notes: Requires Artifactory Pro

Missing values will be set to the default values as defined by the consumed type.

Security: Requires an admin user

Usage: PUT /api/security/users/{userName}
Consumes: application/vnd.org.jfrog.artifactory.security.User+json
Sample Usage:

```
PUT /api/security/users/davids
{
  user.json
}
```

Update User

Description: Updates an existing user in Artifactory with the provided user details.
Since: 2.4.0
Notes: Requires Artifactory Pro
Missing values will be set to the default values as defined by the consumed type
Security: Requires an admin user
Usage: POST /api/security/users/{userName}
Consumes: application/vnd.org.jfrog.artifactory.security.User+json
Sample Usage:

```
POST /api/security/users/davids
{
  user.json
}
```

Delete User

Description: Removes an Artifactory user.
Since: 2.4.0
Notes: Requires Artifactory Pro
Security: Requires an admin user
Usage: DELETE /api/security/users/{userName}
Produces: application/text
Sample Usage:

```
DELETE /api/security/users/davids

User 'davids' has been removed successfully.
```

Expire Password for a Single User

Description: Expires a user's password
Since: 4.4.2
Notes: Requires Artifactory Pro
Security: Requires an admin user
Usage: POST /api/security/users/authorization/expirePassword/{userName}
Sample Usage:

```
POST /api/security/users/authorization/expirePassword/davids
```

Expire Password for Multiple Users

Description: Expires password for a list of users

Since: 4.4.2

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: POST /api/security/users/authorization/expirePassword -H "Content-type: application/json" -d '[{userA}, {userB}]'

Sample Usage:

```
POST /api/security/users/authorization/expirePassword -H "Content-type:
application/json" -d ' [{d Davids}, {johnb}]'
```

Expire Password for All Users

Description: Expires password for all users

Since: 4.4.2

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: POST /api/security/users/authorization/expirePasswordForAllUsers

Sample Usage:

```
POST /api/security/users/authorization/expirePasswordForAllUsers
```

Unexpire Password for a Single User

Description: Unexpires a user's password

Since: 4.4.2

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: POST /api/security/users/authorization/unexpirePassword/{userName}

Produces: application/text

Sample Usage:

```
POST /api/security/users/authorization/unexpirePassword/davids
```

Change Password

Description: Changes a user's password

Since: 4.4.2

Notes: Requires Artifactory Pro

Security: Admin can apply this method to all users, and each (non-anonymous) user can use this method to change his own password.

Usage: POST /api/security/users/authorization/changePassword -H "Content-type: application/json" -d '{ "userName" : "{user}", "oldPassword" : "{old password}", "newPassword1" : "{new password}", "newPassword2" : "{verify new password}" }

Produces: application/text

Sample Usage:

```
POST /api/security/users/authorization/changePassword -H "Content-type:
application/json" -d ' { "userName" : "davids", "oldPassword" : "op",
"newPassword1" : "np", "newPassword2" : "np" }
```

Get Password Expiration Policy

Description: Retrieves the password expiration policy

Since: 4.4.2

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: GET /api/security/configuration/passwordExpirationPolicy

Produces: application/json

Sample Usage:

```
GET /api/security/configuration/passwordExpirationPolicy
{
  "enabled": "true"
  "passwordMaxAge": "60"
  "notifyByEmail": "true"
}
```

Set Password Expiration Policy

Description: Sets the password expiration policy

Since: 4.4.2

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: PUT /api/security/configuration/passwordExpirationPolicy -H "Content-type: application/json" -d '{ "enabled" : "true|false", "passwordMaxAge" : "1-999", "notifyByEmail": "true|false" }

Produces: application/json

Sample Usage:

```
POST /api/security/configuration/passwordExpirationPolicy -H "Content-type:
application/json" -d ' { "enabled" : "true", "passwordMaxAge" : "60",
"notifyByEmail": "true" }
```

Configure User Lock Policy

Description: Configures the user lock policy that locks users out of their account if the number of repeated incorrect login attempts exceeds the configured maximum allowed.

Security: Requires a valid admin user

Usage: PUT /api/security/userLockPolicy

Produces: application/text

Since: 4.4

Sample usage:

```
PUT http://{host}:{port}/artifactory/api/security/userLockPolicy -H
'Content-Type: application/json'-d '
{
  "enabled" : true|false,
  "loginAttempts" : {value}
}'
```

Retrieve User Lock Policy

Description: Retrieves the currently configured user lock policy.

Security: Requires a valid admin user

Usage: GET /api/security/userLockPolicy

Produces: application/json

Since: 4.4

Sample usage:

```
GET http://{host}:{port}/artifactory/api/security/userLockPolicy
'{
  "enabled" : true|false,
  "loginAttempts" : {value}
}'
```

Get Locked Out Users

Description: If locking out users is enabled, lists all users that were locked out due to recurrent incorrect login attempts.

Security: Requires a valid admin user

Usage: GET /api/security/lockedUsers

Produces: application/json

Since: 4.4

Sample Usage:

```
GET /api/security/lockedUsers

[ "usera", "userb", ... ]
```

Unlock Locked Out User

Description: Unlocks a single user that was locked out due to recurrent incorrect login attempts.

Security: Requires a valid admin user

Usage: POST /api/security/unlockUsers/{userName}

Produces: application/text

Since: 4.4

Sample Usage:

```
POST /api/security/unlockUsers/{userName}
```

Unlock Locked Out Users

Description: Unlocks a list of users that were locked out due to recurrent incorrect login attempts.

Security: Requires a valid admin user

Usage: POST /api/security/unlockUsers

Produces: application/text

Since: 4.4

Sample Usage:

```
POST /api/security/unlockUsers -H 'Content-Type: application/json' -d '[
{userA}, {userB} ]'
```

Unlock All Locked Out Users

Description: Unlocks all users that were locked out due to recurrent incorrect login attempts.

Security: Requires a valid admin user

Usage: POST /api/security/unlockAllUsers

Produces: application/text

Since: 4.4

Sample Usage:

```
POST /api/security/unlockAllUsers
```

Create API Key

Description: Create an API key for the current user. Returns an error if API key already exists - use regenerate API key instead.

Since: 4.3.0

Usage: POST /api/security/apiKey

Produces: application/json

Sample input:

```
POST /api/security/apiKey
```

Sample output:

```
{
  "apiKey": "30loposOtVFyCMrT+cXmCAScmVMPrSYXkWIjIyDCXsY="
}
```

Regenerate API Key

Description: Regenerate an API key for the current user

Since: 4.3.0

Usage: PUT /api/security/apiKey

Produces: application/json

Sample input:

```
PUT /api/security/apiKey
```

Sample output:

```
{
  "apiKey": "30loposOtVFyCMrT+cXmCAScmVMPrSYXkWIjIyDCXsY="
}
```

Get API Key

Description: Get the current user's own API key

Since: 4.3.0

Usage: GET /api/security/apiKey

Produces: application/json

Sample usage:

```
GET /api/security/apiKey
```

Sample output:

```
{
  "apiKey": "30l0pos0tVFyCMrT+cXmCASCmVMPrSYXkWIjIyDCXsY="
}
```

Revoke API Key

Description: Revokes the current user's API key

Since: 4.3.0

Usage: DELETE /api/security/apiKey

Produces: application/json

Revoke User API Key

Description: Revokes the API key of another user

Since: 4.3.0

Security: Requires a privileged user (Admin only)

Usage: DELETE /api/security/apiKey/{username}

Produces: application/json

Revoke All API Keys

Description: Revokes all API keys currently defined in the system

Since: 4.3.0

Security: Requires a privileged user (Admin only)

Usage: DELETE /api/security/apiKey?deleteAll={0/1}

Produces: application/json

Get Groups

Description: Get the groups list

Since: 2.4.0

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: GET /api/security/groups

Produces: [application/vnd.org.jfrog.artifactory.security.Users+json](#), [application/vnd.org.jfrog.artifactory.security.Groups+json](#), [application/vnd.org.jfrog.artifactory.security.PermissionTargets+json](#)

Sample Output:

```
GET /api/security/groups
[
  {
    "name": "readers"
    "uri" : "http://localhost:8081/artifactory/api/security/groups/readers"
  }, {
    "name": "tech-leads"
    "uri" :
    "http://localhost:8081/artifactory/api/security/groups/tech-leads"
  }
]
```

Get Group Details

Description: Get the details of an Artifactory Group

Since: 2.4.0

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: GET /api/security/groups/{groupName}

Produces: application/vnd.org.jfrog.artifactory.security.Group+json

Sample Output:

```
GET /api/security/groups/dev-leads
{
  group.json
}
```

Create or Replace Group

Description: Creates a new group in Artifactory or replaces an existing group

Since: 2.4.0

Notes: Requires Artifactory Pro

Missing values will be set to the default values as defined by the consumed type.

Security: Requires an admin user

Usage: PUT /api/security/groups/{groupName}

Consumes: application/vnd.org.jfrog.artifactory.security.Group+json

Sample Usage:

```
PUT /api/security/groups/dev-leads
{
  group.json
}
```

Update Group

Description: Updates an exiting group in Artifactory with the provided group details.

Since: 2.4.0

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: POST /api/security/groups/{groupName}

Consumes: application/vnd.org.jfrog.artifactory.security.Group+json

Sample Usage:

```
POST /api/security/groups/dev-leads
{
  group.json
}
```

Delete Group

Description: Removes an Artifactory group.

Since: 2.4.0

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: DELETE /api/security/groups/{groupName}

Produces: application/text

Sample Usage:

```
DELETE /api/security/groups/dev-leads
```

```
Group 'dev-leads' has been removed successfully.
```

Get Permission Targets

Description: Get the permission targets list

Since: 2.4.0

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: GET /api/security/permissions

Produces: application/vnd.org.jfrog.artifactory.security.Users+json, application/vnd.org.jfrog.artifactory.security.Groups+json, application/vnd.org.jfrog.artifactory.security.PermissionTargets+json

Sample Output:

```
GET /api/security/permissions
[
  {
    "name": "readSourceArtifacts"
    "uri" :
    "http://localhost:8081/artifactory/api/security/permissions/readSourceArtifacts"
  }, {
    "name": "populateCaches"
    "uri" :
    "http://localhost:8081/artifactory/api/security/permissions/populateCaches"
  }
]
```

Get Permission Target Details

Description: Get the details of an Artifactory Permission Target

Since: 2.4.0

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: GET /api/security/permissions/{permissionTargetName}

Produces: application/vnd.org.jfrog.artifactory.security.PermissionTarget+json

Sample Output:

```
GET /api/security/permissions/populateCaches
{
  permission-target.json
}
```

Create or Replace Permission Target

Description: Creates a new permission target in Artifactory or replaces an existing permission target

Since: 2.4.0

Notes: Requires Artifactory Pro

Missing values will be set to the default values as defined by the consumed type.

Security: Requires an admin user

Usage: PUT /api/security/permissions/{permissionTargetName}

Consumes: application/vnd.org.jfrog.artifactory.security.PermissionTarget+json

Sample Usage:

```
PUT /api/security/permissions/populateCaches
{
  permission-target.json
}
```

Delete Permission Target

Description: Deletes an Artifactory permission target.

Since: 2.4.0

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: DELETE /api/security/permissions/{permissionTargetName}

Produces: application/text

Sample usage:

```
DELETE /api/security/permissions/populateCaches

Permission Target 'remoteCachePopulation' has been removed successfully.
```

Effective Item Permissions

Description: Returns a list of effective permissions for the specified item (file or folder).

Only users and groups with some permissions on the item are returned. Supported by local and local-cached repositories.

Permissions are returned according to the following conventions:

m=admin; d=delete; w=deploy; n=annotate; r=read

Notes: Requires Artifactory Pro

Since: 2.3.4

Security: Requires a valid admin or local admin user.

Usage: GET /api/storage/{repoKey}/{itemPath}?permissions

Produces: application/vnd.org.jfrog.artifactory.storage.ItemPermissions+json

Sample Output:

```
GET /api/storage/libs-release-local/org/acme?permissions
{
  "uri":
  "http://localhost:8081/artifactory/api/storage/libs-release-local/org/acme"
  "principals": {
    "users" : {
      "bob": ["r","w","m"],
      "alice": ["d","w","n","r"]
    },
    "groups" : {
      "dev-leads" : ["m","r","n"],
      "readers" : ["r"]
    }
  }
}
```

Security Configuration

Description: Retrieve the security configuration (security.xml).

Since: 2.2.0

Notes: This is an advanced feature - make sure the new configuration is really what you wanted before saving.

Security: Requires a valid admin user

Usage: GET /api/system/security

Produces: application/xml

Sample Output:

```
GET /api/system/security

<security.xml/>
```

Save Security Configuration (Deprecated)

Description: Save the security configuration (security.xml). Requires the security.xml file from the same version.

Since: 2.2.0

Notes: This API is **deprecated**.

Security: Requires a valid admin user

Usage: POST /api/system/security

Consumes: application/xml

Sample Usage:

```
POST /api/system/security

<security.xml/>
```

Activate Master Key Encryption

Description: Creates a new master key and activates master key encryption.

Since: 3.2.2

Notes: This is an advanced feature intended for administrators

Security: Requires a valid admin user

Usage: POST /api/system/encrypt

Produces: text/plain

Sample Usage:

```
POST /api/system/encrypt

DONE
```

Deactivate Master Key Encryption

Description: Removes the current master key and deactivates master key encryption.

Since: 3.2.2

Notes: This is an advanced feature intended for administrators

Security: Requires a valid admin user

Usage: POST /api/system/decrypt

Produces: text/plain

Sample Usage:

```
POST /api/system/decrypt

DONE
```

Set GPG Public Key

Description: Sets the public key that Artifactory provides to Debian and Opkg clients to verify packages

Security: Requires a valid admin user

Usage: PUT /api/gpg/key/public

Produces: text/plain

Since: 3.3

Sample Usage:

```
PUT /api/gpg/key/public
```

Get GPG Public Key

Description: Gets the public key that Artifactory provides to Debian and Opkg clients to verify packages

Security: Requires an authenticated user, or anonymous (if "Anonymous Access" is globally enabled)

Usage: GET /api/gpg/key/public

Produces: text/plain

Since: 3.3

Sample Usage:

```
GET /api/gpg/key/public
```

Set GPG Private Key

Description: Sets the private key that Artifactory will use to sign Debian and ipk packages

Security: Requires a valid admin user

Usage: PUT /api/gpg/key/private

Produces: text/plain

Since: 3.3

Sample Usage:

```
PUT /api/gpg/key/private
```

Set GPG Pass Phrase

Description: Sets the pass phrase required signing Debian and ipk packages using the private key

Security: Requires a valid admin user

Usage: PUT /api/gpg/key/passphrase

Headers: -H X-GPG-PASSPHRASE:passphrase

Produces: text/plain

Since: 3.3

Sample Usage:

```
PUT /api/gpg/key/passphrase
```

Create Token

Description: Creates an access token

Since: 5.0.0

Security: Requires a valid user

Usage: POST /api/security/token

Content-Type: application/x-www-form-urlencoded

Produces: application/json

```
{
  "access_token": "<the access token>",
  "expires_in": <Validity period in seconds>,
  "scope": "<access scope>",
  "token_type": "Bearer",
  "refresh_token": "<the refresh token if access_token is refreshable>"
}
```

Sample Usage:

```
curl -uadmin:password -XPOST
"http://localhost:8081/artifactory/api/security/token" -d "username=johnq"
-d "scope=member-of-groups:readers"

200
{
  "access_token": "adsdgbtybbeeyh...",
  "expires_in": 3600,
  "scope": "api:* member-of-groups:readers",
  "token_type": "Bearer",
  "refresh_token": "fgsfgsdugh8dgu9s8gy9hsg..."
}
```

This endpoint takes the following parameters:

grant_type	[Optional, default: "client_credentials"] The grant type used to authenticate the request. In this case, the only value supported is "client_credentials" which is also the default value if this parameter is not specified.
username	The user name for which this token is created. If the user does not exist, a transient user is created. Non-admin users can only create tokens for themselves so they must specify their own username. If the user does not exist, the member-of-groups scope token must be provided (e.g. member-of-groups: g1, g2, g3...)
scope	[Optional if the user specified in username exists] The scope to assign to the token provided as a space-separated list of scope tokens. Currently there are three possible scope tokens: <ul style="list-style-type: none"> • "api:*" - indicates that the token grants access to REST API calls. This is always granted by default whether specified in the call or not. • member-of-groups:[<group-name>] - indicates the groups that the token is associated with (e.g. member-of-groups: g1, g2, g3...). The token grants access according to the permission targets specified for the groups listed. Specify "*" for group-name to indicate that the token should provide the same access privileges that are given to the group of which the logged in user is a member. A non-admin user can only provide a scope that is a subset of the groups to which he belongs • "jfrt@<instance-id>:admin" - provides admin privileges on the specified Artifactory instance. This is only available for administrators. If omitted and the username specified exists, the token is granted the scope of that user.
expires_in	[Optional, default: 3600] The time in seconds for which the token will be valid. To specify a token that never expires, set to zero. Non-admin can only set a value that is equal to or less than the default 3600.

refreshable	[Optional, default: false] If true, this token is refreshable and the refresh token can be used to replace it with a new token once it expires.
audience	[Optional, default: Only the service ID of the Artifactory instance that created the token] A space-separate list of the other Artifactory instances or services that should accept this token identified by their Artifactory Service IDs as obtained from the Get Service ID endpoint. In case you want the token to be accepted by all Artifactory instances you may use the following audience parameter "audience=jfrrt@*".

Refresh Token

Description: Refresh an access token to extend its validity. If only the access token and the refresh token are provided (and no other parameters), this pair is used for authentication. If username or any other parameter is provided, then the request must be authenticated by a token that grants admin permissions.

Since: 5.0.0

Security: Requires a valid user (unless both access token and refresh token are provided)

Usage: POST /api/security/token

Content-Type: application/x-www-form-urlencoded

Produces: application/json (Please refer to [Create Token](#))

Sample Usage:

```
curl -XPOST "http://localhost:8081/artifactory/api/security/token" -d
"grant_type=refresh_token" -d "refresh_token=fgsg53t3g..." -d
"access_token=gsfdgw35gt..."
```

200 (Success) As in Create Token

400 (Error) If the token was created by a different Artifactory instance
(and hence cannot be refreshed)

This endpoint takes the following parameters:

grant_type	Should be set to <code>refresh_token</code> .
refresh_token	The refresh token of the access token that needs to be refreshed.
access_token	The access token to refresh.
username	Please refer to Create Token . Note: <code>access_token</code> and <code>username</code> are mutually exclusive, so only one of these parameters should be specified.
scope	If <code>access_token</code> is provided, the new token is created with the same settings as that token.
expires_in	
refreshable	
audience	

Revoke Token

Description: Revoke an access token

Since: 5.0.0

Security: Requires a valid user

Usage: POST /api/security/token/revoke

Content-Type: application/x-www-form-urlencoded

Produces: application/json

Sample Usage:

```
curl -uadmin:password -XPOST
"http://localhost:8081/artifactory/api/security/token/revoke" -d
"token=fasdt3..."

200 OK (Also returned if the token was already revoked or non-existent)

400 (Error) If the token was created by a different Artifactory instance
(and hence cannot be revoked)
```

This endpoint takes the following parameters:

token	The token to be revoked
--------------	-------------------------

Get Service ID

Description: Provides the service ID of an Artifactory instance or cluster. Up to version 5.5.1, the Artifactory service ID is formatted `jf-artifactory@<id>`. From version 5.5.2 the service ID is formatted `jfirt@<id>`.

Since: 5.0.0

Security: Requires an admin user

Usage: GET /api/system/service_id

Produces: text/plain

Sample Usage:

```
curl -uadmin:password -XGET
"http://localhost:8081/artifactory/api/system/service_id"

200
jfirt@ee27b1d1-534d-4723-80b5-5bd893d19c43
```

Get Certificates

Description: Returns a list of installed SSL certificates.

Since: 5.4.0

Security: Requires an admin user

Usage: GET /api/system/security/certificates

Produces: application/json

```
[
  {
    "certificateAlias" : "<The Certificate Alias>",
    "issuedTo" : "<The entity to whom the certificate was issued>",
    "issuedBy" : "<The issuing entity>",
    "issuedOn" : "<ISO8601 (yyyy-MM-dd'T'HH:mm:ss.SSSZ)>",
    "validUntil" : "<ISO8601 (yyyy-MM-dd'T'HH:mm:ss.SSSZ)>",
    "fingerPrint" : "<The certificate's SHA256 fingerprint>"
  }
]
```

Sample Usage:

```
GET /api/system/security/certificates

[
  {
    "certificateAlias" : "example1",
    "issuedTo" : "JFrog",
    "issuedBy" : "Some_CA",
    "issuedOn" : "Sun May 01 2017 10:00:00 GMT +02:00 (UTC)",
    "validUntil" : "Sun May 01 2019 10:00:00 GMT +02:00 (UTC)",
    "fingerPrint" : "ab:cd:ef:gh"
  },
  {
    "certificateAlias" : "example2",
    "issuedTo" : "Cool-Company",
    "issuedBy" : "Some_Other_CA",
    "issuedOn" : "Sun May 01 2017 10:00:00 GMT +02:00 (UTC)",
    "validUntil" : "Sun May 01 2019 10:00:00 GMT +02:00 (UTC)",
    "fingerPrint" : "ab:cd:ef:gh"
  }
]
```

Add Certificate

Description: Adds an SSL certificate.

Since: 5.4.0

Security: Requires an admin user

Usage: POST /api/system/security/certificates/{Certificate_alias} -T {Certificate PEM file}

Consumes: application/text

Produces: application/json

```
{
  "status" : 200,
  "message" : ["The certificates were successfully installed"]
}
```

Delete Certificate

Description: Deletes an SSL certificate.

Since: 5.4.0

Security: Requires an admin user

Usage: DELETE /api/system/security/certificates/{Certificate_alias}

Produces: application/json

Sample Usage:

```
DELETE /api/security/certificates/cert1
```

Response:

```
{
  "status" : 200,
  "message" : "The certificates were successfully deleted"
}
```

REPOSITORIES

Get Repositories

Description: Returns a list of minimal repository details for all repositories of the specified type.

Since: 2.2.0

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/repositories[?type=repositoryType (local|remote|virtual|distribution)]

Produces: application/vnd.org.jfrog.artifactory.repositories.RepositoryDetailsList+json

Sample Output:

```
GET /api/repositories
[
  {
    "key" : "libs-releases-local",
    "type" : "LOCAL",
    "description" : "Local repository for in-house libraries",
    "url" : "http://localhost:8081/artifactory/libs-releases-local"
  }, {
    "key" : "libs-snapshots-local",
    "type" : "LOCAL",
    "description" : "Local repository for in-house snapshots",
    "url" : "http://localhost:8081/artifactory/libs-snapshots-local"
  }
]
```

Repository Configuration

Description: Retrieves the current configuration of a repository. Supported by local, remote and virtual repositories.

Since: 2.3.0

Notes: Requires Artifactory Pro

Security: Requires an admin user for complete repository configuration. Non-admin users will receive only partial configuration data.

Usage: GET /api/repositories/{repoKey}

Produces: application/vnd.org.jfrog.artifactory.repositories.LocalRepositoryConfiguration+json, application/vnd.org.jfrog.artifactory.repositories.RemoteRepositoryConfiguration+json, application/vnd.org.jfrog.artifactory.repositories.VirtualRepositoryConfiguration+json

Sample Output:

```
GET /api/repositories/libs-release-local
{
  repository-config.json
}
```

Create Repository

Description: Creates a new repository in Artifactory with the provided configuration. Supported by local, remote and virtual repositories.

Since: 2.3.0

Notes: Requires Artifactory Pro

An existing repository with the same key are removed from the configuration and its content is removed!

Missing values are set to the default values as defined by the consumed type spec.

Security: Requires an admin user

Usage: PUT /api/repositories/{repoKey}

Consumes: application/vnd.org.jfrog.artifactory.repositories.LocalRepositoryConfiguration+json, application/vnd.org.jfrog.artifactory.repositories.RemoteRepositoryConfiguration+json,

application/vnd.org.jfrog.artifactory.repositories.VirtualRepositoryConfiguration+json

Sample Usage:

```
PUT /api/repositories/libs-release-local
{
  repository-config.json
}
```

Update Repository Configuration

Description: Updates an exiting repository configuration in Artifactory with the provided configuration elements. Supported by local, remote and virtual repositories.

Since: 2.3.0

Notes: Requires Artifactory Pro

The class of a repository (the `rclass` attribute cannot be updated.

Security: Requires an admin user

Usage: POST /api/repositories/{repoKey} -H "Content-Type: application/json"

Consumes: application/vnd.org.jfrog.artifactory.repositories.LocalRepositoryConfiguration+json, application/vnd.org.jfrog.artifactory.repositories.RemoteRepositoryConfiguration+json,

application/vnd.org.jfrog.artifactory.repositories.VirtualRepositoryConfiguration+json

Sample Usage:

```
POST /api/repositories/libs-release-local -H "Content-Type:
application/json"
{
  repository-config.json
}
```

Delete Repository

Description: Removes a repository configuration together with the whole repository content. Supported by local, remote and virtual repositories.

Since: 2.3.0

Notes: Requires Artifactory Pro

Security: Requires an admin user

Usage: DELETE /api/repositories/{repoKey}

Produces: application/text

Sample Usage:

```
DELETE /api/repositories/libs-release-local
```

```
Repository 'libs-release-local' and all its content have been removed successfully.
```

Remote Repository Configuration (Deprecated)

Description: Repository Configuration (Deprecated)

Gets the shared configuration of a remote repository.

Since: 2.2.0

Notes: This API is **deprecated**. Use the Get Repository Configuration API instead.

Security: Requires a valid user for a shared remote repository and admin user for anything else. Shared remote repository data will be sanitized for security when non-admin user is used.

Usage: GET /api/repositories/{remoteRepoName}/configuration

Produces: application/vnd.org.jfrog.artifactory.repositories.SharedRemoteRepositoryConfiguration+json

Sample Output:

```
GET /api/repositories/remote-repo/configuration
{
  repository-config.json
}
```

Calculate YUM Repository Metadata

Description: For Local repositories: calculates/recalculates the YUM metadata for this repository, based on the RPM package currently hosted in the repository. Supported by local and virtual repositories only.

Calculation can be synchronous (the default) or asynchronous.

For Virtual repositories, calculates the merged metadata from all aggregated repositories on the specified path. The **path** parameter must be passed for virtual calculation.

Please see the [YUM integration](#) documentation for more details.

Notes: Requires Artifactory Pro. Immediate calculation requests cannot be called on repositories with automatic asynchronous calculations enabled (applies to local repositories only). The **path** parameter applies to virtual repositories only.

Security: Up to version 4.8 , requires a valid admin user. From version 4.8 only requires the set of permissions assumed by Manage (Manage + Delete/Overwrite + Deploy/Cache + Annotate + Read).

Usage: POST /api/yum/{repoKey}[?path={path to repodata dir}][&async=0/1]

Headers (Optionally): -H X-GPG-PASSPHRASE:passphrase

Produces: application/text

Since: 2.3.5

Sample Output:

```
POST /api/yum/yum-local?async=1
POST /api/yum/yum-virtual?path=7/os/x86_64&async=1

YUM metadata calculation for repository 'yum-local' accepted.
```

Calculate NuGet Repository Metadata

Description: Recalculates all the NuGet packages for this repository (local/cache/virtual), and re-annotate the NuGet properties for each NuGet package according to it's internal nuspec file.

Please see the [NuGet integration](#) documentation for more details.

Supported by local, local-cache, remote and virtual repositories.

Notes: Requires Artifactory Pro.

Security: Up to version 4.8 , requires a valid admin user. From version 4.8 only requires the set of permissions assumed by Manage (Manage + Delete/Overwrite + Deploy/Cache + Annotate + Read).

Usage: POST /api/nuget/{repoKey}/reindex

Produces: application/text

Since: 3.0.3

Sample Output:

```
POST /api/nuget/nuget-local/reindex
```

```
NuGet reindex calculation for repository 'nuget-local' accepted.
```

Calculate Npm Repository Metadata

Description: Recalculates the npm search index for this repository (local/virtual). Please see the [Npm integration](#) documentation for more details. Supported by local and virtual repositories.

Notes: Requires Artifactory Pro.

Security: Up to version 4.8 , requires a valid admin user. From version 4.8 only requires the set of permissions assumed by Manage (Manage + Delete/Overwrite + Deploy/Cache + Annotate + Read).

Usage: POST /api/npm/{repoKey}/reindex

Produces: application/text

Since: 3.2.0

Sample Output:

```
POST /api/npm/npm-local/reindex
```

```
Recalculating index for npm repository npm-local scheduled to run
```

Calculate Maven Index

Description: Calculates/caches a Maven index for the specified repositories.

For a virtual repository specify all underlying repositories that you want the aggregated index to include.

Calculation can be forced, which for remote repositories will cause downloading of a remote index even if a locally cached index has not yet expired; and index recalculation based on the cache on any failure to download the remote index, including communication errors (the default behavior is to only use the cache when a remote index cannot be found and returns a 404). Forcing has no effect on local repositories index calculation.

Please see the [Exposing Maven Indexes](#) documentation for more details.

Notes: Requires Artifactory Pro.

Security: Up to version 4.8 , requires a valid admin user. From version 4.8 only requires the set of permissions assumed by Manage (Manage + Delete/Overwrite + Deploy/Cache + Annotate + Read).

Usage: POST /api/maven[?repos=x[,y]][&force=0/1]

Produces: application/text

Since: 2.5.0

Sample Output:

```
POST /api/maven?repos=libs-release-local,ext-release-local&force=1
```

```
Maven index refresh for repositories '[libs-release-local,
ext-release-local]' has been accepted.
```

Calculate Maven Metadata

Description: Calculates Maven metadata on the specified path (local repositories only).

Security: Up to version 4.8 , requires a valid admin user. From version 4.8 only requires the set of permissions assumed by Manage (Manage + Delete/Overwrite + Deploy/Cache + Annotate + Read).

Usage: POST /api/maven/calculateMetadata/{repoKey}/{folder-path}?[nonRecursive=true | false]

Produces: application/text

Since: 3.0.2

Sample Output:

```
POST /api/maven/calculateMetadata/libs-release-local/org/acme
OK
```

Calculate Debian Repository Metadata

Description: Calculates/recalculates the Packages and Release metadata for this repository, based on the Debian packages in it. Calculation can be synchronous (the default) or asynchronous. Please refer to [Debian Repositories](#) for more details. Supported by local repositories only.

From version 4.4, by default, the recalculation process also writes several entries from the Debian package's metadata as properties on all of the artifacts (based on the control file's content).

This operation may not always be required (for example, if the Debian files are intact and were not modified, only the index needs to be recalculated. The operation is resource intensive and can be disabled by passing the `?writeProps=0` query param.

Notes: Requires Artifactory Pro.

Security: Up to version 4.8, requires a valid admin user. From version 4.8 only requires the set of permissions assumed by Manage (Manage + Delete/Overwrite + Deploy/Cache + Annotate + Read).

Usage: `POST api/deb/reindex/{repoKey} [?async=0/1][?writeProps=0/1]`

Headers (Optionally): `-H X-GPG-PASSPHRASE:passphrase`

Produces: application/text

Since: 3.3

Sample Output:

```
POST /api/deb/reindex/debian-local

Recalculating index for Debian repository debian-local scheduled to run.
```

Calculate Opkg Repository Metadata

Description: Calculates/recalculates the Packages and Release metadata for this repository, based on the ipk packages in it (in each feed location).

Calculation can be synchronous (the default) or asynchronous. Please refer to [Opkg Repositories](#) for more details. Supported by local repositories only.

By default, the recalculation process also writes several entries from the ipk package's metadata as properties on all of the artifacts (based on the control file's content).

This operation may not always be required (for example, if the ipk files are intact and were not modified, only the index needs to be recalculated. The operation is resource intensive and can be disabled by passing the `?writeProps=0` query param.

Notes: Requires Artifactory Pro.

Security: Up to version 4.8, requires a valid admin user. From version 4.8 only requires the set of permissions assumed by Manage (Manage + Delete/Overwrite + Deploy/Cache + Annotate + Read).

Usage: `POST api/opkg/reindex/{repoKey} [?async=0/1][?writeProps=0/1]`

Headers (Optionally): `-H X-GPG-PASSPHRASE:passphrase`

Produces: application/text

Since: 4.4

Sample Output:

```
POST /api/opkg/reindex/opkg-local

Recalculating index for Opkg repository opkg-local scheduled to run.
```

Calculate Bower Index

Description: Recalculates the index for a Bower repository.

Notes: Requires Artifactory Pro.

Security: Up to version 4.8, requires a valid admin user. From version 4.8 only requires the set of permissions assumed by Manage (Manage + Delete/Overwrite + Deploy/Cache + Annotate + Read).

Usage: `POST api/bower/{repoKey}/reindex`

Produces: application/text

Since: 3.6.0

Sample Output:

```
POST /api/bower/bower-local/reindex

Bower index for refresh for bower-local has been accepted
```

SYSTEM & CONFIGURATION

System Info

Description: System Info

Get general system information.

Since: 2.2.0

Security: Requires a valid admin user

Usage: GET /api/system

Produces: text/plain

Sample Output:

```
GET /api/system

system info output text
```

System Health Ping

Description: Get a simple status response about the state of Artifactory

Returns 200 code with an 'OK' text if Artifactory is working properly, if not will return an HTTP error code with a reason.

Since: 2.3.0

Security: Requires a valid user (can be anonymous). If artifactory.ping.allowUnauthenticated=true is set in artifactory.system.properties, then no authentication is required even if anonymous access is disabled.

Usage: GET /api/system/ping

Produces: text/plain

Sample Output:

```
GET /api/system/ping

OK
```

Verify Connection

Description: Verifies a two-way connection between Artifactory and another product

Returns Success (200) if Artifactory receives a similar success code (200) from the provided endpoint. See possible error codes below.

Since: 4.15.0

Security: Requires an admin user.

Usage: POST/api/system/verifyconnection

Consumes: application/json

```
POST /api/system/verifyconnection
{
  + "endpoint" : "<The URL that Artifactory should connect to>",
  - "username" : "<Username to be used for the connection test>",
  - "password" : "<Password to be used for the connection test>"
}
```

Produces: application/json

Upon success (200):
200 Successfully connected to endpoint

Upon error, returns 400 along with a JSON object that contains the error returned from the other system.

Sample Output:

```
{
  "errors" : [ {
    "status" : 400,
    "message" : "Received error from endpoint url: HTTP 404: Page not
found"
  } ]
}
```

General Configuration

Description: Get the general configuration (artifactory.config.xml).

Since: 2.2.0

Security: Requires a valid admin user

Usage: GET /api/system/configuration

Produces: application/xml (http://www.jfrog.org/xsd/artifactory-v1_7_3.xsd)

Sample Output:

```
GET /api/system/configuration

<artifactory.config.xml/>
```

Save General Configuration

Description: Save the general configuration (artifactory.config.xml).

Since: 2.2.0

Notes: This is an advanced feature - make sure the new configuration is really what you wanted before saving.

Security: Requires a valid admin user

Usage: POST /api/system/configuration

Consumes: application/xml (http://www.jfrog.org/xsd/artifactory-v1_7_3.xsd)

Sample Usage:

```
POST /api/system/configuration

<artifactory.config.xml/>
```

Update Custom URL Base

Description: Changes the Custom URL base

Since: 3.9.0

Security: Requires a valid admin user

Usage: PUT /api/system/configuration/baseUrl

Example: curl -X PUT "http://localhost:8081/artifactory/api/system/configuration/baseUrl" -d 'https://mycompany.com:444/artifactory' -uadmin:password -H "Content-type: text/plain"

Sample Output:

```
URL base has been successfully updated to
"https://mycompany.com:444/artifactory".
```

License Information

Description: Retrieve information about the currently installed license.

Since: 3.3.0

Security: Requires a valid admin user

Usage: GET /api/system/license

Produces: application/json

Sample Output:

```
GET /api/system/license
{
  "type" : "Commercial",
  "validThrough" : "May 15, 2014",
  "licensedTo" : "JFrog inc."
}
```

Install License

Description: Install new license key or change the current one.

Since: 3.3.0

Security: Requires a valid admin user

Usage: POST /api/system/license

Produces: application/json

Consumes: application/json ({ "licenseKey": "your supplied license key ..." })

Sample Output:

```
POST /api/system/license
{
  "status" : 200,
  "message" : "The license has been successfully installed."
}
```

HA License Information

Description: Retrieve information about the currently installed licenses in an HA cluster.

Since: 5.0.0

Security: Requires a valid admin user

Usage: GET /api/system/licenses

Produces: application/json

```
[ {
  "type" : "Enterprise",
  "validThrough" : "<validity date formatted MMM DD, YYYY>",
  "licensedTo" : "<Company name>",
  "licenseHash" : "<license hash code>",
  "nodeId" : "<Node ID of the node activated with this license | Not in
use>",
  "nodeUrl" : "<URL of the node activated with this license | Not in
use>",
  "expired" : <true | false>
}]
```

Sample Output:

```
GET /api/system/licenses
```

```
[ {
  "type" : "Enterprise",
  "validThrough" : "May 15, 2018",
  "licensedTo" : "JFrog",
  "licenseHash" : "179b7ea384d0c4655a00dfac7285a21d986a17923",
  "nodeId" : "art1",
  "nodeUrl" : "http://10.1.16.83:8091/artifactory",
  "expired" : false
}, {
  "type" : "Enterprise",
  "validThrough" : "May 15, 2018",
  "licensedTo" : "JFrog",
  "licenseHash" : "e10b8aa1d1dc5107439ce43debc6e65dfef71afd3",
  "nodeId" : "Not in use",
  "nodeUrl" : "Not in use",
  "expired" : false
} ]
```

Install HA Cluster Licenses

Description: Install a new license key(s) on an HA cluster.

Since: 5.0.0

Security: Requires an admin user

Usage: POST /api/system/licenses

Consumes: application/json

```
[{
  "licenseKey": "<License key>"
}]
```

Produces: application/json


```
{
  "status" : 200,
  "messages" : {
    ["<License key>" : "<status message>"]
  }
}
```

Sample Usage:

```
POST /api/system/licenses
[
  {
    "licenseKey": "tL9r2Y...lDBiktbbt"
  },
  {
    "licenseKey": DiYgVA...P7nvyNI7q"
  }
]
```

Response:

```
{
  "status" : 200,
  "messages" : {
    "tL9r2Y...lDBiktbbt" : "OK",
    "DiYgVA...P7nvyNI7q" : "OK",
  }
}
```

Delete HA Cluster License

Description: Deletes a license key from an HA cluster.

Since: 5.0.0

Security: Requires an admin user

Usage: DELETE /api/system/licenses?licenseHash=licenseHash1, licenseHash2...

Produces: application/json

```
{
  "status" : 200,
  "messages" : [{"<License hash code>" : "<status message>"}]
}
```

Sample Usage:

```
DELETE /api/system/licenses?licenseHash=tL9r2YlDBiktbbt, DiYgVAP7nvyNI7q
```

Response:

```
{
  "status" : 200,
  "messages" : {
    "tL9r2YlDBiktbbt" : "OK",
    "DiYgVAP7nvyNI7q" : "OK"
  }
}
```

Version and Add-ons information

Description: Retrieve information about the current Artifactory version, revision, and currently installed Add-ons

Since: 2.2.2

Security: Requires a valid user (can be anonymous)

Usage: GET /api/system/version

Produces: application/vnd.org.jfrog.artifactory.system.Version+json

Sample Output:

```
GET /api/system/version
{
  "version" : "2.2.2",
  "revision" : "10427",
  "addons" : [ "build", "ldap", "properties", "rest", "search", "sso",
    "watch", "webstart" ]
}
```

Get Reverse Proxy Configuration

Description: Retrieves the reverse proxy configuration

Since: 4.3.1

Security: Requires a valid admin user

Usage: GET /api/system/configuration/webServer

Produces: application/json

Sample Output:

```
GET /api/system/configuration/webServer

{
  "key" : "nginx",
  "webServerType" : "NGINX",
  "artifactoryAppContext" : "artifactory",
  "publicAppContext" : "artifactory",
  "serverName" : "jfrog.com",
  "serverNameExpression" : "*.jfrog.com",
  "artifactoryServerName" : "localhost",
  "artifactoryPort" : 8081,
  "sslCertificate" : "/etc/ssl/myKey.cert",
  "sslKey" : "/etc/ssl/myKey.key",
  "dockerReverseProxyMethod" : "SUBDOMAIN",
  "useHttps" : true,
  "useHttp" : true,
  "sslPort" : 443,
  "httpPort" : 76
}
```

Update Reverse Proxy Configuration

Description: Updates the reverse proxy configuration

Since: 4.3.1

Security: Requires an admin user

Usage: POST /api/system/configuration/webServer

Consumes: application/json

Sample Usage:

```
POST /api/system/configuration/webServer

{
  "key" : "nginx",
  "webServerType" : "NGINX",
  "artifactoryAppContext" : "artifactory",
  "publicAppContext" : "artifactory",
  "serverName" : "jfrog.com",
  "serverNameExpression" : "*.jfrog.com",
  "artifactoryServerName" : "localhost",
  "artifactoryPort" : 8081,
  "sslCertificate" : "/etc/ssl/myKey.cert",
  "sslKey" : "/etc/ssl/myKey.key",
  "dockerReverseProxyMethod" : "SUBDOMAIN",
  "useHttps" : true,
  "useHttp" : true,
  "sslPort" : 443,
  "httpPort" : 76
}
```

Get Reverse Proxy Snippet

Description: Gets the reverse proxy configuration snippet in text format

Since: 4.3.1

Security: Requires a valid user (not anonymous)

Usage: GET /api/system/configuration/reverseProxy/nginx

Produces: text/plain

Sample Usage:

```
GET /api/system/configuration/reverseProxy/nginx

## add ssl entries when https has been set in config
ssl_certificate      /etc/ssl/myKey.cert;
ssl_certificate_key  /etc/ssl/myKey.key;
ssl_session_cache   shared:SSL:1m;
ssl_prefer_server_ciphers  on;
## server configuration
server {
    listen 443 ssl;
    listen 76 ;
    server_name ~(?<repo>.+)\.jfrog.com jfrog.com;

    if ($http_x_forwarded_proto = '') {
        set $http_x_forwarded_proto $scheme;
    }
    ## Application specific logs
    ## access_log /var/log/nginx/jfrog.com-access.log timing;
    ## error_log /var/log/nginx/jfrog.com-error.log;
    rewrite ^/$ /artifactory/webapp/ redirect;
    rewrite ^/artifactory$ /artifactory/webapp/ redirect;
}
```

Create Bootstrap Bundle

Description: This rest is relevant for High Availability set up. It will create a bootstrap bundle on the primary node of an Artifactory HA installation that will include all the relevant keys so a new node can access the database and fetch all the relevant configuration files. The same bundle must be installed on all nodes during an installation of new nodes or if upgrading from a version older than 5.0. For more details, please refer to [Installing Artifactory HA](#).

Since: 5.0.0

Security: Requires an admin user

Usage: POST /api/system/bootstrap_bundle

Produces: application/json

```
{
  "file" : "<Location on primary node where bootstrap bundle was created>"
}
```

Sample usage:

```
POST /api/system/bootstrap_bundle

{
  "file" : "/opt/jfrog/artifactory/etc/bootstrap.bundle.tar.gz"
}
```

PLUGINS

Execute Plugin Code

Description: Executes a named execution closure found in the `executions` section of a **user plugin**. Execution can take parameters and be synchronous (the default) or asynchronous.

When parameters can have multiple values, you can separate the items in one of the following ways:

- Use a semicolon - ; (recommended)
- Use the encoding for the pipe ("|") character - %7C
Alternatively, you may configure your NGINX to encode URLs so that if an unencoded pipe is used in the URL, NGINX will encode it to %7C. We recommend that you verify that this configuration does not break any other systems served by NGINX

Since: 2.3.1

Notes: Requires Artifactory Pro

Security: Requires an authenticated user (the plugin can control which users/groups are allowed to trigger it)

Usage: POST /api/plugins/execute/{executionName}?[params=p1=v1[,v2][|p2=v3][&async=1]]

Produces: text/plain

Sample Output:

```
POST
/api/plugins/execute/cleanup?params=suffix=SNAPSHOT;types=jar,war,zip&asyn
c=1
OK
```

Retrieve Plugin Code

Description: Retrieves the source code of the specified user plugin.

Since: 5.0.0

Notes: Requires Artifactory Pro

Security: Requires an admin user.

Usage: GET /api/plugins/download/{pluginName}

Produces: text/x-groovy-source

Sample Usage

```
GET /api/plugins/download/myPlugin

Response:
<The source code of the plugin>
```

Retrieve Plugin Info

Description: Retrieves **user plugin** information for **Executions** and **Staging** plugins (subject to the permissions of the provided credentials).

Since: 2.5.2

Notes: Requires Artifactory Pro

Security: Requires an authenticated user.

Usage: GET /api/plugins

Produces: application/json

Sample Output:

```
GET /api/plugins
{
  "executions": [
    {
      "name": "execution1",
      "version": "version",
      "description": "description",
      "users": ["user1"],
      "groups": ["group1", "group2"],
      "params": {}
    }
  ],
  "staging": [
    {
      "name": "strategy1",
      "version": "1.0",
      "description": "desc",
      "params": {"key1": "vall"}
    }
  ]
}
```

Retrieve Plugin Info Of A Certain Type

Description: Retrieves all available **user plugin** information (subject to the permissions of the provided credentials) of the specified type.

Since: 2.5.2

Notes: Requires Artifactory Pro

Security: Requires an authenticated user.

Usage: GET /api/plugins/{pluginType}

Produces: application/json

Sample Output:

```
GET /api/plugins/staging
{
  "staging": [
    {
      "name": "strategy1",
      "version": "1.0",
      "description": "desc",
      "params": {"key1": "vall"}
    }
  ]
}
```

Retrieve Build Staging Strategy

Description: Retrieves a build staging strategy defined by a **user plugin**.

When passing in parameters that may take multiple values, you can separate the items in one of the following ways:

- Use a semicolon - ; (recommended)
- Use the encoding for the pipe ("|") character - %7C

Alternatively, you may configure your NGINX to encode URLs so that if an unencoded pipe is used in the URL, NGINX will encode it to %7C. We recommend that you verify that this configuration does not break any other systems served by NGINX

Since: 2.5.2

Notes: Requires Artifactory Pro

Security: Requires an authenticated user.

Usage: GET /api/plugins/build/staging/{strategyName}?buildName={buildName}&[params=p1=v1[,v2][[p2=v3]]

Produces: application/vnd.org.jfrog.plugins.BuildStagingStrategy

Sample Output:

```
GET
/api/plugins/build/staging/strategy1?buildName=build1&params=types=jar,war
,zip
{
  "defaultModuleVersion":
  {
    "moduleId": "moduleId",
    "nextRelease": "nextRelease",
    "nextDevelopment": "nextDevelopment"
  },
  "vcsConfig":
  {
    "useReleaseBranch": true,
    "releaseBranchName": "branchName",
    "createTag": true,
    "tagUrlOrName": "tagUrl",
    "tagComment": "comment",
    "nextDevelopmentVersionComment": "comment"
  },
  "promotionConfig":
  {
    "targetRepository": "repoKey",
    "comment": "comment",
    "status": "statusName"
  }
}
```

Execute Build Promotion

Description: Executes a named promotion closure found in the `promotions` section of a **user plugin**.

Since: 2.5.2

Notes: Requires Artifactory Pro

Security: Requires an authenticated user.

Usage: POST /api/plugins/build/promote/{promotionName}/{buildName}/{buildNumber}?[params=p1=v1[,v2][[p2=v3]]

Produces: text/plain

Sample Output:

```
POST
/api/plugins/build/promote/promotion1/build1/3?params=types=jar,war,zip
OK
```

Reload Plugins

Description: Reloads user plugins if there are modifications since the last user plugins reload. Works regardless of the automatic user plugins

refresh interval.

Since: 2.9.0

Notes: Requires Artifactory Pro

Security: Requires a valid admin user

Usage: POST /api/plugins/reload

Produces: text/plain

Sample Output:

```
POST /api/plugins/reload
Successfully loaded: myplugin1.groovy, myplugin2.groovy
```

IMPORT & EXPORT

Import Repository Content

Description: Import one or more repositories.

Since: 2.2.2

Security: Requires a valid admin user

Usage: POST: /api/import/repositories

Requests Params:

path - The file system path to import from. This may point to a specific folder to import data for a single repository, or to the parent "repositories" folder to import data for all repositories.

repo - Empty/null repo -> all

metadata - Include metadata - default 1

verbose - Verbose - default 0

Produces: text/plain

Sample Output:

```
POST: /api/import/repositories?path=pathToRepos&verbose=1
```

Import System Settings Example

Description: Returned default Import Settings JSON.

Since: 2.4.0

Security: Requires a valid admin user

Usage: GET: /api/import/system

Produces: application/vnd.org.jfrog.artifactory.system.ImportSettings+json

Sample Usage:

```
GET /api/import/system
{
  "importPath" : "/import/path",
  "includeMetadata" : true,
  "verbose" : false,
  "failOnError" : true,
  "failIfEmpty" : true
}
```

Full System Import

Description: Import full system from a server local Artifactory export directory.

Since: 2.4.0

Security: Requires a valid admin user
Usage: POST: /api/import/system
Consumes : application/vnd.org.jfrog.artifactory.system.ImportSettings+json
Produces: text/plain
Sample Usage:

```
POST /api/import/system
{
  import-settings.json
}
```

Export System Settings Example

Description: Returned default Export Settings JSON.
Since: 2.4.0
Security: Requires a valid admin user
Usage: GET: /api/export/system
Produces: application/vnd.org.jfrog.artifactory.system.ExportSettings+json
Sample Usage:

```
GET /api/export/system
{
  "exportPath" : "/export/path",
  "includeMetadata" : true,
  "createArchive" : false,
  "bypassFiltering" : false,
  "verbose" : false,
  "failOnError" : true,
  "failIfEmpty" : true,
  "m2" : false,
  "incremental" : false,
  "excludeContent" : false
}
```

Export System

Description: Export full system to a server local directory.
Since: 2.4.0
Security: Requires a valid admin user
Usage: POST: /api/export/system
Consumes : application/vnd.org.jfrog.artifactory.system.ExportSettings+json, application/json
Produces: text/plain
Sample Usage:

```
POST /api/export/system{ export-settings.json }
```

SUPPORT

Create Bundle

Description: Create a new support bundle.
Since: 4.3.0

Security: Requires an admin user
Notes: All bundle items are optional.
Usage: POST /api/support/bundles/
Sample Usage:

```

POST /api/support/bundles
{
  - "systemLogsConfiguration" : {
    "enabled" : true,(default)
    "startDate" : {date-in-millis},
    "endDate" : {date-in-millis}
  },
  - "systemInfoConfiguration" : {
    "enabled" : true (default)
  },
  - "securityInfoConfiguration" : {
    "enabled" : true, (default)
    "hideUserDetails" : true (default)
  },
  - "configDescriptorConfiguration" : {
    "enabled" : true, (default)
    "hideUserDetails" : true (default)
  },
  - "configFilesConfiguration" : {
    "enabled" : true (default),
    "hideUserDetails" : true (default)
  },
  - "storageSummaryConfiguration" : {
    "enabled" : true (default)
  },
  - "threadDumpConfiguration" : {
    "enabled" : true, (default)
    "count" : {amount-of-dumps},          (1 is default)
    "interval" : {interval in millis} (0 is default)
  }
}

```

NOTE: systemLogsConfiguration parameter can also be expressed as number of days as follows:

```

...
- "systemLogsConfiguration" : {
  "enabled" : true,(default)
  "daysCount" : {number-of-days}
},
...

{
  "bundles" : [
    "/api/support/bundles/support-bundle-20151118-1709272-1447859367247.zip"
  ]
}

```

Description: Lists previously created bundle currently stored in the system.

Since: 4.3.0

Security: Requires a privileged user (Admin only)

Usage: GET /api/support/bundles/

Produces: application/json

Sample Usage:

```
GET /api/support/bundles

{
  "bundles" : [
    "/api/support/bundles/support-bundle-20151118-1709272-1447859367247.zip",
    "/api/support/bundles/support-bundle-20151117-1035500-1447749350025.zip",
    "/api/support/bundles/support-bundle-20151117-1035147-1447749314704.zip"
  ]
}
```

Get Bundle

Description: Downloads a previously created bundle currently stored in the system.

Since: 4.3.0

Security: Requires a privileged user (Admin only)

Usage: GET /api/support/bundles/{bundle-name}

Produces: application/json

Sample Usage:

```
GET /api/support/bundles/support-bundle-20151122-1705472-1448211947280.zip
```

Delete Bundle

Description: Deletes a previously created bundle from the system.

Since: 4.3.0

Security: Requires a privileged user (Admin only)

Usage: DELETE /api/support/bundles/{bundle-name}

Produces: application/json

Sample Usage:

```
DELETE
/api/support/bundles/support-bundle-20151122-1705472-1448211947280.zip
```

ERROR RESPONSES

In case of an error, Artifactory will return an error response in JSON format. The response contains the HTTP status code and error message.

For example, a badly formatted API call would return the "404, File not found" response below:

```
{
  "errors" : [ {
    "status" : 404,
    "message" : "File not found."
  } ]
}
```

Sample input:

```
POST /api/security/apiKey
{
  "apiKey": "30loposOtVFyCMrT+cXmCASCmVMPPrSYXkWIjIyDCXsY="
}
```

```
PUT /api/storage/libs-release-local/ch/qos/logback/logback-classic/0.9.9?properties=os=win,linux;qa=done&recursive=1
```

When parameters can have multiple values, you can separate the items in one of the following ways:

- Use a semicolon - ; (recommended)
- Use the encoding for the pipe ("|") character - %7C
Alternatively, you may configure your NGINX to encode URLs so that if an unencoded pipe is used in the URL, NGINX will encode it to %7C. We recommend that you verify that this configuration does not break any other systems served by NGINX

Repository Configuration JSON

Repository Configuration JSON

Legend

+	Mandatory element in create/replace queries (optional in "update" queries)
-	Optional element in create/replace queries
(default)	The default value when unspecified in create/replace queries

Page Contents

- Repository Configuration JSON
- Local Repository
 - application/vnd.org.jfrog.artifactory.repositories.LocalRepositoryConfiguration+json
- Remote Repository
 - application/vnd.org.jfrog.artifactory.repositories.RemoteRepositoryConfiguration+json
- Virtual Repository
 - application/vnd.org.jfrog.artifactory.repositories.VirtualRepositoryConfiguration+json

Local Repository

application/vnd.org.jfrog.artifactory.repositories.LocalRepositoryConfiguration+json

```

{
  - "key": "local-repo",
  + "rclass" : "local",
  - "packageType": "maven" | "gradle" | "ivy" | "sbt" | "nuget" | "gems"
  | "npm" | "bower" | "debian" | "composer" | "pypi" | "docker" |
  "vagrant" | "gitlfs" | "yum" | "conan" | "chef" | "puppet" | "generic"
  (default)
  - "description": "The local repository public description",
  - "notes": "Some internal notes",
  - "includesPattern": "**/*" (default),
  - "excludesPattern": "" (default),
  - "repoLayoutRef" : "maven-2-default",
  - "debianTrivialLayout" : false,
  - "checksumPolicyType": "client-checksums" (default) |
  "server-generated-checksums"
  - "handleReleases": true (default),
  - "handleSnapshots": true (default),
  - "maxUniqueSnapshots": 0 (default),
  - "maxUniqueTags": 0 (default)
  - "snapshotVersionBehavior": "unique" | "non-unique" (default) |
  "deployer",
  - "suppressPomConsistencyChecks": false (default),
  - "blackedOut": false (default),
  - "propertySets": ["ps1", "ps2"],
  - "archiveBrowsingEnabled" : false,
  - "calculateYumMetadata" : false,
  - "yumRootDepth" : 0,
  - "dockerApiVersion" : "V2" (default),
  - "enableFileListsIndexing " : "false" (default)
}

```

Remote Repository

application/vnd.org.jfrog.artifactory.repositories.RemoteRepositoryConfiguration+json

```
{
  - "key": "remote-repo1",
  + "rclass" : "remote",
  - "packageType": "maven" | "gradle" | "ivy" | "sbt" | "nuget" | "gems"
  | "npm" | "bower" | "debian" | "pypi" | "docker" | "yum" | "vcs" |
  "composer" | "p2" | "chef" | "puppet" | "generic" (default)
  + "url" : "http://host:port/some-repo",
  - "username": "remote-repo-user",
  - "password": "pass",
  - "proxy": "proxyl",
  - "description": "The remote repository public description",
  - "notes": "Some internal notes",
  - "includesPattern": "**/*" (default),
  - "excludesPattern": "" (default),
  - "repoLayoutRef" : "maven-2-default",
  - "remoteRepoChecksumPolicyType": "generate-if-absent" (default) |
  "fail" | "ignore-and-generate" | "pass-thru",
  - "handleReleases": true (default),
  - "handleSnapshots": true (default),
  - "maxUniqueSnapshots": 0 (default),
  - "suppressPomConsistencyChecks": false (default),
  - "hardFail": false (default),
  - "offline": false (default),
  - "blackedOut": false (default),
  - "storeArtifactsLocally": true (default),
  - "socketTimeoutMillis": 15000 (default),
  - "localAddress": "212.150.139.167",
  - "retrievalCachePeriodSecs": 43200 (default),
  - "failedRetrievalCachePeriodSecs": 30 (default),
  - "missedRetrievalCachePeriodSecs": 7200 (default),
  - "unusedArtifactsCleanupEnabled": false (default),
  - "unusedArtifactsCleanupPeriodHours": 0 (default),
  - "assumedOfflinePeriodSecs" : 300 (default),
  - "fetchJarsEagerly": false (default),
  - "fetchSourcesEagerly": false (default),
  - "shareConfiguration": false (default),
  - "synchronizeProperties": false (default),
  - "blockMismatchingMimeTypes" : true (default),
  - "propertySets": ["ps1", "ps2"],
  - "allowAnyHostAuth": false (default),
  - "enableCookieManagement": false (default),
  - "bowerRegistryUrl": "https://bower.herokuapp.com" (default),
  - "vcsType": "GIT" (default),
  - "vcsGitProvider": "GITHUB" (default) | "BITBUCKET" | "OLDSTASH" |
  "STASH" | "ARTIFACTORY" | "CUSTOM",
  - "vcsGitDownloadUrl": "" (default),
  - "bypassHeadRequest" : false (default)
  - "clientTlsCertificate": "" (default)
}
```


Virtual Repository

application/vnd.org.jfrog.artifactory.repositories.VirtualRepositoryConfiguration+json

```
{
  - "key": "virtual-repo1",
  + "rclass" : "virtual",
  + "packageType": "maven" | "gradle" | "ivy" | "sbt" | "nuget" | "gems"
  | "npm" | "bower" | "pypi" | "docker" | "p2" | "yum" | "chef" | "puppet"
  | "generic"
  - "repositories": ["local-rep1", "local-rep2", "remote-rep1",
"virtual-rep2"]
  - "description": "The virtual repository public description",
  - "notes": "Some internal notes",
  - "includesPattern": "**/*" (default),
  - "excludesPattern": "" (default),
  - "debianTrivialLayout" : false
  - "artifactoryRequestsCanRetrieveRemoteArtifacts": false,
  - "keyPair": "keypair1",
  - "pomRepositoryReferencesCleanupPolicy": "discard_active_reference"
(default) | "discard_any_reference" | "nothing"
  - "defaultDeploymentRepo": "local-repo1"
}
```

Security Configuration JSON

Security Configuration JSON

Legend

+	Mandatory element in create/replace queries, optional in "update" queries
-	Optional element in create/replace queries
!	Read-only element
(default)	The default value when unspecified in create/replace queries

application/vnd.org.jfrog.artifactory.security.User+json

```

{
  - "name": "davids",
  + "email" : "davids@jfrog.com",
  + "password": "****" (write-only, never returned),
  - "admin": false (default),
  - "profileUpdatable": true (default),
  - "disableUIAccess" : true,
  - "internalPasswordDisabled": false (default),
  ! "lastLoggedIn": ISO8601 (yyyy-MM-dd'T'HH:mm:ss.SSSZ),
  ! "realm": "Internal",
  - "groups" : [ "deployers", "users" ]
}

```

application/vnd.org.jfrog.artifactory.security.Group+json

```

{
  - "name": "dev-leads",
  - "description" : "The development leads group",
  - "autoJoin" : false (default, must be false if adminPrivileges is true),
  - "adminPrivileges" : false (default),
  - "realm": "Realm name (e.g. ARTIFACTORY, CROWD)",
  - "realmAttributes": "Realm attributes for use by LDAP"
}

```

application/vnd.org.jfrog.artifactory.security.PermissionTarget+json

Permissions are set/returned according to the following conventions:
m=admin; d=delete; w=deploy; n=annotate; r=read

name - limited to 64 characters

includePattern/excludePattern - limited to 1024 characters

```

{
  - "name": "populateCaches",
  - "includesPattern": "*" (default),
  - "excludesPattern": "" (default),
  + "repositories": ["local-rep1", "local-rep2", "remote-rep1",
"virtual-rep2"],
  - "principals": {
    "users" : {
      "bob": ["r","w","m"],
      "alice" : ["d","w","n", "r"]
    },
    "groups" : {
      "dev-leads" : ["m","r","n"],
      "readers" : ["r"]
    }
  }
}

```

System Settings JSON

System Settings JSON

Legend

+	Mandatory element
-	Optional element
(default)	The default value when unspecified

application/vnd.org.jfrog.artifactory.system.ImportSettings+json

```
{
  + "importPath" : "/import/path" (A path to a directory on the local file
system of Artifactory server),
  - "includeMetadata" : true (default),
  - "verbose" : false (default),
  - "failOnError" : true (default),
  - "failIfEmpty" : true (default)
}
```

application/vnd.org.jfrog.artifactory.system.ExportSettings+json

```
{
  + "exportPath" : "/export/path" (A path to a directory on the local file
system of Artifactory server),
  - "includeMetadata" : true (default),
  - "createArchive" : false (default),
  - "bypassFiltering" : false (default),
  - "verbose" : false (default),
  - "failOnError" : true (default),
  - "failIfEmpty" : true (default),
  - "m2" : false (default),
  - "incremental" : false (default),
  - "excludeContent" : false (default)
}
```

application/vnd.org.jfrog.artifactory.system.Version+json

```
{
  "version" : "2.2.2",
  "revision" : "10427",
  "addons" : [ "build", "ldap", "properties", "rest", ... ] (list of active
addons)
}
```

Configuring Artifactory

Overview

You can access the General Configuration settings of Artifactory in the **Admin** tab under **Configuration | General**.

Saving changes

Any changes you make must be saved in order for them to take effect.

General Settings

The fields under General Settings allow you to set up various global parameters in Artifactory. The ? icon next to each field provides a detailed description of the field. Fields marked with a red star are mandatory.

General Configuration

General Settings

Server Name ?

Custom URL Base ?

File Upload Max Size * ?

Bintray Max Files Upload *

Date Format * ?

Global Offline Mode ?

Enable Help Component

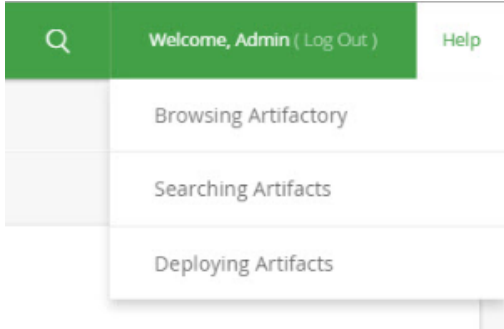
Page Contents

- [Overview](#)
- [General Settings](#)
- [Global Replication Blocking](#)
- [Folder Download Settings](#)
- [Trash Can Settings](#)
- [Look and Feel Settings \(Branding\)](#)
- [Custom Message](#)

Read More

- [Configuring the Database](#)
- [Configuring the Filestore](#)
- [Checksum-Based Storage](#)
- [Configuring Repositories](#)
- [Configuring Security](#)
- [Configuring a Reverse Proxy](#)
- [Mail Server Configuration](#)
- [Configuration Files](#)
- [Exposing Maven Indexes](#)
- [Clustering Artifactory](#)

The General Settings fields are as follows:

Server Name (mandatory)	The name of the server to be displayed on the title of each page.
Custom URL Base	<p>By default, URLs generated in Artifactory use the context URL returned by your servlet container as a base. A custom URL base is useful when Artifactory is running behind a proxy. In this case the base for URLs generated in Artifactory for links and redirect responses must be specified manually.</p> <p>Another reason to change the base URL would be to have non-request-based operations in Artifactory use the correct address, for example in generated emails.</p> <p>This may also be modified using the Update Custom URL Base REST API.</p> <div style="border: 1px solid green; padding: 5px;"><p>Overriding the Custom URL Base You can override the Custom URL Base by adding an <code>X-Artifactory-Override-Base-Url</code> HTTP header to your request. This can be useful if you need Artifactory to answer to more than one host name.</p></div>
File Upload Max Size	Maximum size allowed for files uploaded via the web interface.
Bintray Max Files Upload	The maximum number of files that can be pushed to Bintray in a single operation.
Date Format (mandatory)	The date format for displaying dates in the web interface.
Global Offline Mode	When checked, Artifactory will behave as if it is not connected to an external network (such as the internet), and therefore, will not query remote repositories (regardless of the offline status of any specific remote repository).
Enable Help Component	<p>If set, Artifactory will display context sensitive help topics in the top right corner of the UI.</p> 

Global Replication Blocking

By configuring Global Replication Blocking, you can enable or disable replication globally as needed.

Global Replication Blocking

- Block Replications [?](#)
- Block Push Replications [?](#)
- Block Pull Replications [?](#)

Block Replications	When set, push and pull replication will be blocked according to the check boxes below, regardless of the configuration for specific repositories.
Block Push Replications	When set, push replication will be blocked regardless of the configuration for specific repositories.
Block Pull Replications	When set, pull replication will be blocked regardless of the configuration for specific repositories.

Folder Download Settings

Folder Download Settings

Enable Folder Download

Max Size [?](#)

1024

Max Number of Files [?](#)

5000

Max Parallel Folder Downloads [?](#)

10

Enable Folder Download	Must be set to enable folder download
Max Size	The maximum size (in MB) of a folder that may be downloaded
Max Number of Files	The maximum number artifacts that may be downloaded under one folder
Max Parallel Folder Downloads	The maximum number of folder download requests that may be run concurrently


Trash Can Settings

Trash Can Settings

Enable Trash Can

Retention Period (Days) ?

30

 Empty Trashcan

Enable Trash Can	If set, trash can will be enabled and deleted items will be stored in the trash can for the specified retention period.
Retention Period	The number of days to keep deleted items in the trash can before deleting permanently.
Empty Trash Can	Click to delete all items currently in the trash can permanently.

Look and Feel Settings (Branding)

In the Look and Feel Settings you can configure Artifactory to present your company logo in the top left corner of the screen and customize the footer text.


You may either upload an image to be used locally, or reference a remote image URL.

Look and Feel Settings

Upload from: **File** Url

Logo File

Drop file here or **Select File** Clear



File	If selected, upload your company logo image file.
	<div style="border: 1px solid orange; padding: 5px;"> <p>Logo file upload The uploaded logo file is copied into the <code>ARTIFACTORY_HOME/etc/ui</code> folder.</p> </div>
URL	If selected, specify the URL from which Artifactory should display your company logo

Custom Message

You can display a **Custom Message** at the top of Artifactory screens.

Upgrade Notice: We have upgraded to Artifactory 4. Make sure all your repositories are [single package type](#).

To configure the custom message, in the **Admin** module, select **Configuration | General**.

Custom Message

Enabled

Show Only in Home Page

Title

Upgrade Notice

Title Color

#43A047

Message

```
We have upgraded to Artifactory 4. Make sure all your repositories are
[http://www.jfrog.com/confluence/display/RTF/Upgrading+Artifactory#UpgradingArtifactory-SinglePackageTypeRepositories,single package type].
```

 To embed a link inside the message use : [http://example.com, text]

Enabled	Toggles the custom message on and off.
Show Only in Home Page	When set, the custom message will only be displayed on the Artifactory Home Page .
Title	The title for the message.
Title Color	Click on the colored rectangle to select a color, or enter the color in Hex format.
Message	The message to display. You can include links in the message using the following format: [<link URL>, <link text>]

Configuring the Database

Overview

Artifactory comes with a built-in embedded Derby database that can be reliably used to store data (metadata) for production-level repositories up to hundreds of gigabytes in size.

However, Artifactory supports pluggable database implementations allowing you to change the default to use other popular databases.

Artifactory currently supports the following databases:

- Derby (The default embedded database)
- MySQL v5.5 and above with InnoDB
- Oracle version 10g and above
- Microsoft SQL Server 2008 and above
- PostgreSQL v9.2 and above

For each of the supported databases you can find the corresponding properties file inside `$ARTIFACTORY_HOME/misc/db`.

Choosing the Right Database

As the default database, Derby provides good performance since it runs in the same process as Artifactory, however, under intensive usage or high load, performance may be degraded since Artifactory and the database compete for shared JVM resources such as caches and memory. Therefore, for Artifactory servers that need to support heavy load, you may consider using an external database such as MySQL or PostgreSQL which are very common choices in many Artifactory installations.

Any of the other supported databases is also a fair choice and may be the practical choice to make if your organization is already using one of them.

Accessing a Remote Database

When using an external database, you need a reliable, stable and low-latency network connection to ensure proper functioning of your system.

When using a `fullDB` configuration, we strongly recommend a high-bandwidth to accommodate the transfer of large BLOBs over the network.

Modes of Operation

Artifactory supports two modes of operation:

- Metadata in the database and binaries stored on the file system (This is the default and recommended configuration).
- Metadata and binaries stored as BLOBs in the database

Checksum-Based Storage

Artifactory uniquely stores artifacts using checksum-based storage. For details, please refer to [Checksum-Based Storage](#).

Page Contents

- [Overview](#)
 - [Choosing the Right Database](#)
 - [Modes of Operation](#)
 - [Checksum-Based Storage](#)
- [Before You Start](#)
 - [Backup Your Current Installation](#)
- [Setup the New Database](#)
 - [Advanced Settings](#)

Read more

- [MySQL](#)

- Oracle
- Microsoft SQL Server
- PostgreSQL

Before You Start

Preprocessing

Changing the database does not automatically transfer your data to the new database. Please follow the steps below to backup your data so that you can restore it after the change.

Backup Your Current Installation

When changing the database for an existing installation you must first perform a [Full System Export](#) using the "**Exclude Content**" option. Once your new database is set up and configured, you will import this data to re-populate your Artifactory metadata content.

Make sure to [backup](#) your current Artifactory system before updating to a new database. You will need your Artifactory instance to be disconnected from the network to avoid usage during this procedure.

Setup the New Database

To setup your new database you need to perform the following steps:

- Create a database instance
- Create an Artifactory user for the database
- Install the appropriate JDBC driver
- Copy the relevant database configuration file
- Configure the corresponding `db.properties` file.
- Start Artifactory
- Import the metadata using [Full System Import](#)

These steps are fully detailed in the specific documentation page for each of the supported databases listed in the [Overview](#).

Advanced Settings

Once you have setup your database, you can configure it to support your expected load with the following two parameters:

pool.max.active	The maximum number of pooled database connections (default: 100).
pool.max.idle	The maximum number of pooled idle database connections (default: 10).

MySQL

Overview

By using MySQL you can benefit from features in the MySQL infrastructure such as backup, restore and high availability.

For Artifactory to run with MySQL you must create a dedicated MySQL database instance and then configure Artifactory to use it as described in the following sections.

Before You Continue

Before proceeding with the steps below, please make sure you have read and followed the steps described in [Configuring the Database](#).

Page Contents

- Overview
- Creating the Artifactory MySQL Database
- Increasing MySQL Default Packet Size
- Tuning MySQL for Artifactory
- Configuring Artifactory to use MySQL

Creating the Artifactory MySQL Database

Supported MySQL Versions

Artifactory supports MySQL v5.5 and above with InnoDB engine which is the default provided.

Artifactory provides a script that will execute the SQL commands you need to create your MySQL database.

The script can be found in `$ARTIFACTORY_HOME/misc/db/createdb/createdb_mysql.sql` and is displayed below.

You should review the script and modify it as needed to correspond to your specific environment.

createdb.sql Script

```
CREATE DATABASE artdb CHARACTER SET utf8 COLLATE utf8_bin;  
GRANT ALL on artdb.* TO 'artifactory'@'localhost' IDENTIFIED BY 'password';  
FLUSH PRIVILEGES;
```

Selecting a Case-Sensitive Collation

While MySQL Database Server is not case-sensitive by default, it is important to select a case-sensitive collation because Artifactory is case-sensitive.

For example, in the `createdb.sql` script above COLLATE is set to "utf8_bin".

Artifactory privileges

We recommend providing Artifactory with full privileges on the database

Increasing MySQL Default Packet Size

Since some data files (builds, configurations etc.) are stored in MySQL, it is extremely important to increase the maximum allowed packet size used by MySQL to avoid errors related to large packets.

(Please refer to MySQL documentation: [Packet Too Large](#)).

It is recommended to change this in the `/etc/my.cnf` file as follows:

Modifying /etc/my.cnf

```
# The MySQL server
[mysqld]
.
# The maximum size of the binary payload the server can handle
max_allowed_packet=8M
.
```

/etc/my.cnf Absolute Path

If */etc/my.cnf* does not already exist it should be created under the absolute path and not under *\$ARTIFACTORY_HOME*

Restart required

After modifying the maximum allowed packed size you need to restart MySQL

You can also use the command line

You can also change the `max_allowed_packet` variable on the MySQL command line as in the following example:

```
SET GLOBAL max_allowed_packet = 134217728;
```

Note, however, that upon a restart, the value of the `max_allowed_packet` variable will be read from the */etc/my.cnf* file and revert to the value in that file as described above.

Tuning MySQL for Artifactory

When using Artifactory with MySQL it is recommended to use the InnoDB engine with the following tuning parameters configured in the */etc/my.cnf* file:

Tuning Parameters for MySQL

```
# The MySQL server
[mysqld]
.
# By default innodb engine use one file for all databases and tables. We
recommend changing this to one file per table.
# NOTE: This will take effect only if Artifactory tables are not created
yet! Need to be set and MySQL restarted before starting Artifactory for the
first time.
innodb_file_per_table

# These are tuning parameters that can be set to control and increase the
memory buffer sizes.
innodb_buffer_pool_size=1536M
tmp_table_size=512M
max_heap_table_size=512M

# These control the innodb log files size and can be changed only when
MySQL is down and MySQL will not start if there are some innodb log files
left in the datadir.
# So, changing theses means removing the old innodb log files before start.
innodb_log_file_size=256M
innodb_log_buffer_size=4M
.
```

Restart required

After tuning, you need to restart MySQL

Configuring Artifactory to use MySQL

1. Copy `$ARTIFACTORY_HOME/misc/db/mysql.properties` to `$ARTIFACTORY_HOME/etc/db.properties` (If you do not have this file you can take it from the standalone zip distribution or directly from the [JFrog domain](#)). For a full explanation on the contents of this file please refer to [The Bundled Storage Configurations](#).
2. Adjust the connection definitions in the `$ARTIFACTORY_HOME/etc/db.properties` file to match the attributes of the Artifactory database you created.
You must configure the database URL and username/password to use. The schema and tables are created first time Artifactory is run using the new database.
3. Download the MySQL JDBC driver (available from the [MySQL website](#)) and copy the `mysql-connector-java-<version>.jar` file into the server's shared lib directory.
For example `$TOMCAT_HOME/lib` when installed as a service or `$ARTIFACTORY_HOME/tomcat/lib` in the standalone version.

Permissions

Make sure your driver has the same permissions as the rest of the files in the shared lib directory.

4. Start Artifactory.

Storing BLOBs inside MySQL

The suggested (and recommended) configuration stores all artifact information in MySQL while the artifact binary data is stored on the

file system (under `$ARTIFACTORY_HOME/data/filestore`).

While it is **not recommended**, it is possible to store BLOBs inside MySQL provided that the typical BLOB size is relatively small. Storing large BLOBs in MySQL can cause memory issues because MySQL buffers BLOBs rather than streaming them (please refer to [MySQL Bug #15089](#)) and may result in `OutOfMemory` errors with large binaries depending on your JVM heap size.

To store BLOBs in MySQL, in the `storage.properties` file set `binary.provider.type=fullDb` and change `max_allowed_packet` to be higher than the maximum artifact size you intend to store in Artifactory.

Oracle

Overview

By using Oracle you can benefit from features in Oracle infrastructure such as backup, restore and high availability.

For Artifactory to run with Oracle you must create a dedicated Oracle database instance and then configure Artifactory to use it as described in the following sections.

Before You Continue

Before proceeding with the steps below, please make sure you have read and followed the steps described in [Configuring the Database](#).

Upgrading the Database?

To avoid a regression of performance while upgrading the Oracle database (as a result of changes in the execution plans), make sure to preserve the optimizer's behavior from the previous version. For more details, please refer to Oracle documentation on [Influencing the Optimizer](#).

Page Contents

- [Overview](#)
- [Creating the Artifactory Oracle Database](#)
- [Configuring Artifactory to use Oracle](#)

Creating the Artifactory Oracle Database

Supported Oracle Versions

Artifactory supports Oracle v10g and above.

You can choose between two configurations to set up your Oracle Database

1. **DB-Filesystem**
This configuration stores metadata in Oracle Database and artifact binary data is stored on the file system (under `$ARTIFACTORY_HOME/data/filestore`). This option has the advantage of being very lightweight on the Oracle database.
2. **Full DB**
This configuration stores both metadata and BLOBs in Oracle Database. This option requires minimal maintenance and allows you to rely solely on Oracle for failover and backup procedures, since all data is in the database. When using this option, make sure you have created a table space big enough to accommodate your binaries.

Artifactory privileges

Artifactory creates all tables automatically first time it is run. When performing a software upgrade Artifactory may have to alter tables

and indices, so make sure you grant the configured connection the appropriate user permissions to perform such actions.

Recommendation

With both of the above options (Full DB and DB-Filesystem), it is recommended to create a dedicated table space and use AL32 UTF8 encoding.

Reclaiming BLOB space

For efficiency, Artifactory uses a checksum to ensure that only one copy of any binary data is stored, however, you may want to reclaim deleted BLOB space from time to time by shrinking the BLOB table space as follows:

Reclaiming Deleted BLOB Space

```
{schema}.binary_blobs modify lob (data) (shrink space cascade);
```

Configuring Artifactory to use Oracle

1. Copy `$ARTIFACTORY_HOME/misc/db/oracle.properties` to `$ARTIFACTORY_HOME/etc/db.properties` (If you do not have this file you can take it from the standalone zip distribution or directly from the [JFrog domain](#)). For a full explanation on the contents of this file please refer to [The Bundled Storage Configurations](#).
2. Adjust the connection definitions in the `$ARTIFACTORY_HOME/etc/db.properties` file to match the attributes of the Artifactory database you created.
You must configure the database URL and username/password to use. The schema and tables are created first time Artifactory is run using the new database.
3. Download the JDBC driver corresponding to your Oracle version from the [JDBC/UCP Download Page](#) and copy the `ojdbc6.jar` file into the server's shared lib directory.
For example `$TOMCAT_HOME/lib` when installed as a service or `$ARTIFACTORY_HOME/tomcat/lib` in the standalone version.

Permissions

Make sure your driver has the same permissions as the rest of the files in the shared lib directory.

4. Start Artifactory.

Microsoft SQL Server

Overview

By using MicrosoftSQL you can benefit from features in the Microsoft SQL Server infrastructure such as backup, restore and high availability.

Optimizing Artifactory when running with MS SQL Server

When running Artifactory with Microsoft SQL Server you may create the Artifactory schema on an existing server used for other applications, however for optimal performance, we recommend creating a dedicated Microsoft SQL Server database instance and then configure Artifactory to use it as described in the following sections.

Before You Continue

Before proceeding with the steps below, please make sure you have read and followed the steps described in [Configuring the Database](#).

Page Contents

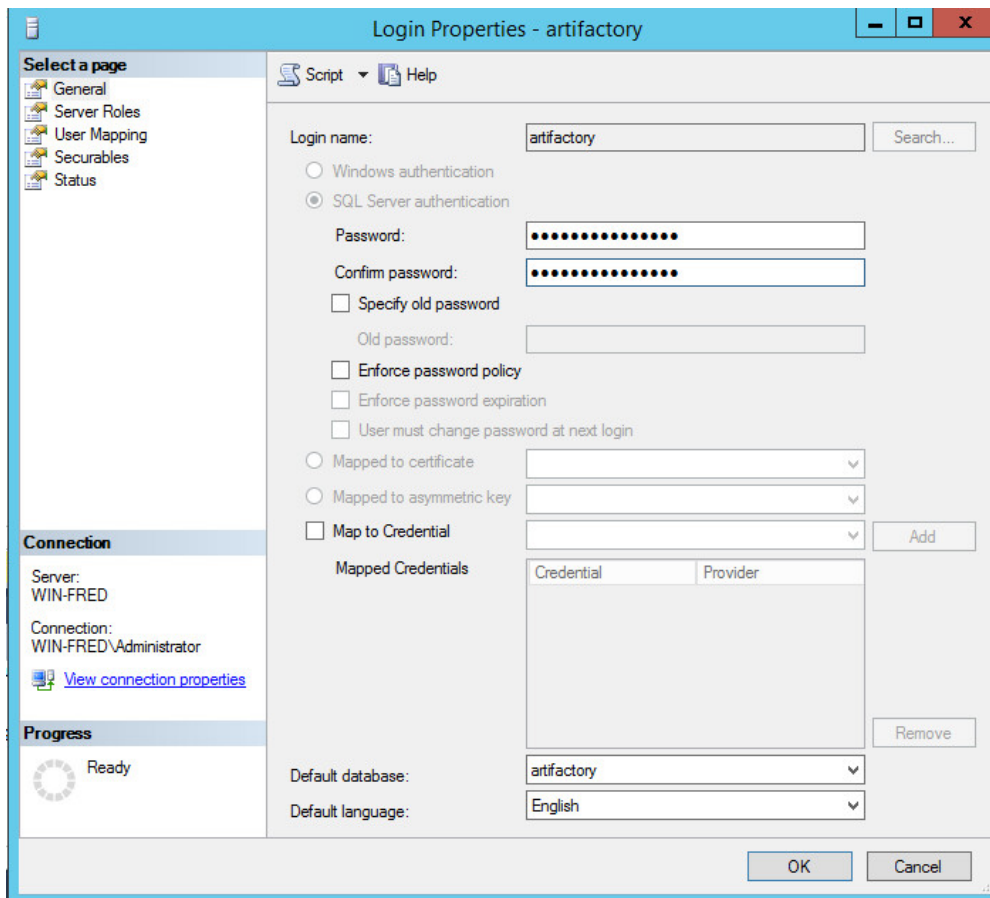
- Overview
- Creating the Artifactory Microsoft SQL Server Database
- Configuring Artifactory to use Microsoft SQL Server

Creating the Artifactory Microsoft SQL Server Database

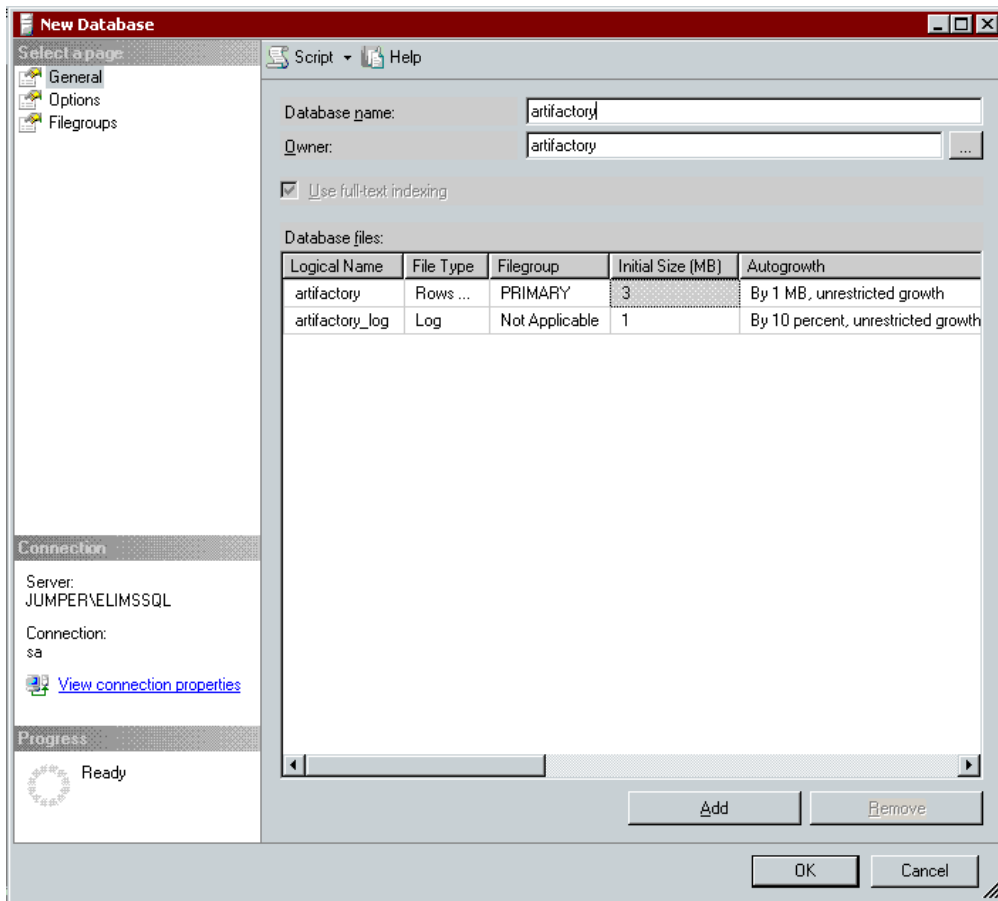
Supported Microsoft SQL Server Versions

Artifactory supports Microsoft SQL Server 2008 and above.

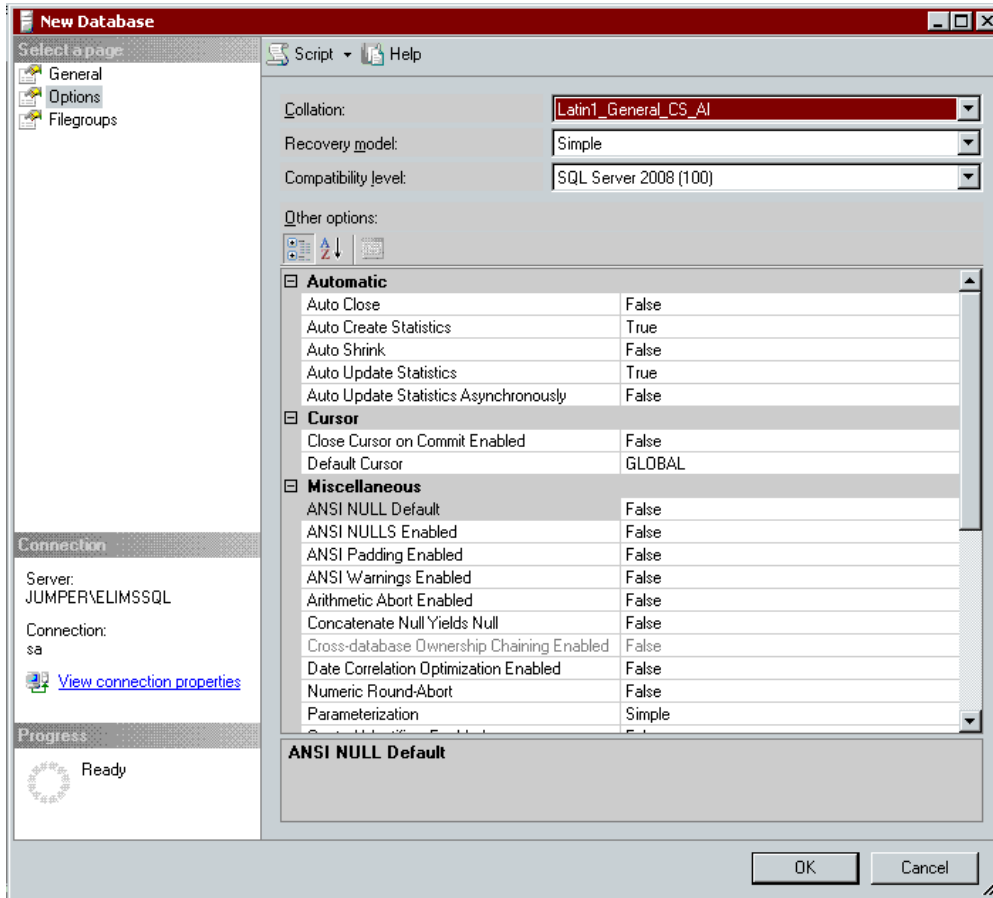
1. Create a new user for Artifactory:
In **Microsoft SQL Server Management Studio**, open the **Object Explorer**, right click on **Security** and select **New | Login...**
2. Create user "artifactory" and set its password.



3. Create the Artifactory database:
In **Microsoft SQL Server Management Studio**, open the **Object Explorer**, right click on **Databases** and select **New database...**
4. In the **New Database** dialog window, select **General** in the **Select a page:** navigation pane.
Set **Database name** to "artifactory" and **Owner** to "artifactory" (the user name you created in step 2).



5. Select the **Options** page and set **Collation** to "Latin1_General_CS_AI". Then click **OK** to confirm.



Selecting a Case-sensitive Collation
 While Microsoft SQL Database Server is not case-sensitive by default, it is important to select a case-sensitive collation because Artifactory is case-sensitive.

Configuring Artifactory to use Microsoft SQL Server

1. Copy `$ARTIFACTORY_HOME/misc/db/mssql.properties` to `$ARTIFACTORY_HOME/etc/db.properties` (If you do not have this file you can take it from the standalone zip distribution or directly from the [JFrog domain](#)). For a full explanation on the contents of this file please refer to [The Bundled Storage Configurations](#).
2. Adjust the connection definitions in the `$ARTIFACTORY_HOME/etc/db.properties` file to match the attributes of the Artifactory database you created. You must configure the database URL and username/password to use. The schema and tables are created first time Artifactory is run using the new database. For example:

Configuring the Database URL and user/password

```
url=jdbc:sqlserver://hostname:1433;databaseName=dbname;sendStringParametersAsUnicode=false;applicationName=Artifactory Binary Repository
```

Where `hostname` is your database address, `1433` is your database port (if not the default 1433), `dbname` is the name of the database you created in the previous step.

sendStringParameterAsUnicode
 Make sure not to overwrite `sendStringParametersAsUnicode=false` since this is critical for appropriate and efficient use of the database indices.

- Download and extract the [Microsoft JDBC Driver](#) and copy the `sqljdbc4.jar` file into the server's shared lib directory. For example `$TOMCAT_HOME/lib` when installed as a service or `$ARTIFACTORY_HOME/tomcat/lib` in the standalone version.

Permissions

Make sure your driver has the same permissions as the rest of the files in the shared lib directory.

- Start Artifactory.

PostgreSQL

Overview

By using PostgreSQL you can benefit from features in PostgreSQL infrastructure such as backup, restore and high availability.

For Artifactory to run with PostgreSQL you must create a dedicated PostgreSQL database instance and then configure Artifactory to use it as described in the following sections.

Before You Continue

Before proceeding with the steps below, please make sure you have read and followed the steps described in [Configuring the Database](#).

Page Contents

- [Overview](#)
- [Creating the Artifactory PostgreSQL Database](#)
- [Configuring Artifactory to use PostgreSQL](#)

Creating the Artifactory PostgreSQL Database

Supported PostgreSQL Versions

Artifactory supports PostgreSQL 9.2 and above using driver version 9.2-1002.jdbc4 and above.

The commands below create artifactory user and database with appropriate permissions.

Use the commands below to create an Artifactory user and database with appropriate permissions. Modify the relevant values to match your specific environment:

Creating an Artifactory User and Database

```
CREATE USER artifactory WITH PASSWORD 'password';
CREATE DATABASE artifactory WITH OWNER=artifactory ENCODING='UTF8';
GRANT ALL PRIVILEGES ON DATABASE artifactory TO artifactory;
```

Artifactory privileges

We recommend providing Artifactory with full privileges on the database.

Configuring Artifactory to use PostgreSQL

1. Copy `$ARTIFACTORY_HOME/misc/db/postgresql.properties` to `$ARTIFACTORY_HOME/etc/db.properties` (If you do not have this file you can take it from the standalone zip distribution or directly from the JFrog domain). For a full explanation on the contents of this file please refer to [The Bundled Storage Configurations](#).
2. Adjust the connection definitions in the `$ARTIFACTORY_HOME/etc/db.properties` file to match the attributes of the Artifactory database you created.
You must configure the database URL and username/password to use. The schema and tables are created first time Artifactory is run using the new database.
3. Download the JDBC driver corresponding to your PostgreSQL version from the [PostgreSQL JDBC Driver Download site](#) and copy the `postgresql-9.x-xxx.jdbc4.jar` file into the server's shared lib directory.
For example `$TOMCAT_HOME/lib` when installed as a service or `$ARTIFACTORY_HOME/tomcat/lib` in the standalone version.

Permissions

Make sure your driver has the same permissions as the rest of the files in the shared lib directory.

4. Start Artifactory.

Storing BLOBs inside PostgreSQL is not recommended

The above recommended configuration keeps all artifact information in PostgreSQL while storing the artifact binary data on the file system (under `$ARTIFACTORY_HOME/data/filestore`).

While it is possible, to store BLOBs inside PostgreSQL **we do not recommend it**. This is important because the PostgreSQL driver doesn't support streaming BLOBs with unknown length to the database. Therefore, Artifactory will temporarily save deployed files to the filesystem and only then save the BLOB to the database.

Configuring the Filestore

Overview

JFrog Artifactory offers flexible filestore management that is configurable to meet a variety of needs in terms of binary storage providers, storage size, and redundancy. Not only are you now able to use different storage providers, but you can also chain a series of providers together to build complex structures of binary providers and support seamless and unlimited growth in storage.

Artifactory offers flexible filestore management through the `binarystore.xml` configuration file located in the `$ARTIFACTORY_HOME/etc` folder. By modifying this file you can implement a variety of different binary storage configurations.

Take care when modifying binarystore.xml

Making changes to this file may result in losing binaries stored in Artifactory!

If you are not sure of what you are doing, please contact [JFrog Support](#) for assistance.

Chains and Binary Providers

The `binarystore.xml` file specifies a chain with a set of binary providers. A binary provider represents a type of object storage feature such as "cached filesystem". Binary providers can be embedded into one another to form chains that represent a coherent filestore. Artifactory comes with a **built-in set of chains** that correspond to the `binary.provider.type` parameter that was used in previous versions of Artifactory. The built-in set of chains available in Artifactory are:

- file-system
- cache-fs
- full-db
- full-db-direct

Page contents

- [Overview](#)
 - [Chains and Binary Providers](#)
- [Configuring a Built-in Filestore](#)
- [Basic Configuration Elements](#)
 - [Built-in Templates](#)
- [Modifying an Existing Filestore](#)

- s3
- google-storage
- double-shards
- redundant-shards
- cluster-file-system
- cluster-s3
- cluster-google-storage

Configuring a Built-in Filestore

To configure Artifactory to use one of the built-in filestores, you only need some basic configuration elements.

- Built-in Chain Templates
 - Filesystem Binary Provider
 - Cached Filesystem Binary Provider
 - Full-DB Binary Provider
 - Full-DB-Direct Binary Provider
 - Cloud Storage Providers
 - S3
 - GCS
 - Azure
 - EFS
 - FS
 - Double Shards, Redundant Shards
 - DFS
 - FS
 - S3
 - Configuring Sharding for HA Cluster
 - FS
 - S3
 - GCS
 - Azure
 - S3
 - FS
- Configuring a Custom Filestore From Scratch
- Configuring the Filestore for Older Artifactory Versions

Basic Configuration Elements

For basic filestore configuration, the *binarystore.xml* file is quite simple and contains the basic tags or elements that are described below along with the attributes that they may include:

config tag

The `<config>` tag specifies a filestore configuration. It includes a `version` attribute to allow versioning of configurations.

```
<config version="v1">
...
</config>
```

chain element

The config tag contains a `chain` element that defines the structure of the filestore. To use one of the built-in filestores, the chain element needs to include the corresponding template attribute. For example, to use the built-in basic "file system" template, all you need is the following configuration:

```
<config version="v1">
  <chain template="file-system"/>
</config>
```

Built-in Templates

The following sections describe the basic chain templates come built-in with Artifactory and are ready for you to use out-of-the-box, as well as other binary providers that are included in the default chains.

Additional information about every template can be found below, under the Built-in Chain Templates section.

file-system	The most basic filestore configuration for Artifactory used for a local or mounted filestore.
cache-fs	Works the same way as filesystem but also has a binary LRU (Least Recently Used) cache for download requests. Improves performance of instances with high IOPS (I/O Operations) or slow NFS access.
full-db	All the metadata and the binaries are stored as BLOBs in the database with an additional layer of caching.
full-db-direct	All the metadata and the binaries are stored as BLOBs in the database without caching.
s3	This is the setting used for S3 Object Storage using the JetS3t library.
s3Old	This is the setting used for S3 Object Storage using JCloud as the underlying framework.
google-storage	This is the setting used for Google Cloud Storage as the remote filestore.
azure-blob-storage	This is the setting used for Azure Blob Storage as the remote filestore.

double-shards	A pure sharding configuration that uses 2 physical mounts with 1 copy (which means each artifact is saved only once).
redundant-shards	A pure sharding configuration that uses 2 physical mounts with 2 copies (which means each shard stores a copy of each artifact).
cluster-file-system	A filestore configuration where each node has its own local filestore (just like the file-system chain) and is connected to all other nodes via dynamically allocated Remote Binary Providers using the Sharding-Cluster provider.
cluster-s3	This is the setting used for S3 Object Storage using the JetS3t library. It is based on the sharding and dynamic provider logic that synchronizes the cluster-file-system.
cluster-google-storage	This is the setting used for Google Cloud Storage using the JetS3t library. It is based on the sharding and dynamic provider logic that synchronizes the cluster-file-system.
cluster-azure-blob-storage	This is the setting used for Azure Blob Storage . It is based on the sharding and dynamic provider logic that synchronizes the cluster-file-system.

Modifying an Existing Filestore

To accommodate any specific requirements you may have for your filestore, you may modify one of the existing chain templates either by extending it with additional binary providers or by overriding one of its attributes. For example, the built-in filesystem chain template stores binaries under the `$(ARTIFACTORY_HOME)/data/filestore` directory. To modify the template so that it stores binaries under `$(FILESTORE)/binaries` you could extend it as follows:

```

<!-- file-system chain template structure -->
<config version="v1">
  <chain template="file-system"/>
  <provider id="file-system" type="file-system" <!-- Modify the
"file-system" binary provider -->
    <baseDataDir>$(FILESTORE)/binaries</baseDataDir> <!-- Override the
<baseDataDir> attribute -->
  </provider>
</config>

```

Built-in Chain Templates

Artifactory comes with a set of chain templates built-in allowing you to set up a variety of different filestores out-of-the-box. However, to override the built-in filestores, you need to be familiar with the attributes available for each binary provider that is used in them. These are described in the following sections which also show the template configuration and what is 'under the hood' in each template. Also, usage examples can be found for all templates.

Filesystem Binary Provider

This is the basic filestore configuration for Artifactory and is used for a local or mounted filestore.

file-system template configuration

If you choose to use the `file-system` template, your `binarystore.xml` configuration file should look like this:

```
<config version="v1">
  <chain template="file-system"/>
</config>
```

What's in the template?

While you don't need to configure anything else in your *binarystore.xml*, this is what the *file-system* template looks like under the hood. In this example, the filestore and temp folder are located under the root directory of the machine.

```
<config version="v1">
  <chain template="file-system"/>
  <provider id="file-system" type="file-system">
    <baseDataDir>/var/opt/jfrog/data</baseDataDir>
    <fileStoreDir>/filestore</fileStoreDir>
    <tempDir>/temp</tempDir>
  </chain>
</provider>
</config>
```

Where:

type	file-system
baseDataDir	Default: <i>\$ARTIFACTORY_HOME/data</i> The root directory where Artifactory should store data files.
fileStoreDir	Default: <i>filestore</i> The root folder of binaries for the filestore. If the value specified starts with a forward slash (" <i>/</i> ") the value is considered the fully qualified path to the filestore folder. Otherwise, it is considered relative to the baseDataDir .
tempDir	Default: <i>temp</i> A temporary folder under baseDataDir into which files are written for internal use by Artifactory. This must be on the same disk as the fileStoreDir .

Cached Filesystem Binary Provider

The *cache-fs* serves as a binary LRU (Least Recently Used) cache for all upload/download requests. This can improve Artifactory's performance since frequent requests will be served from the *cache-fs* (as in case of the S3 binary provider).

The *cache-fs* binary provider will be the closest filestore layer of Artifactory. This means that if the filestore is mounted, we would like the *cache-fs* to be local on the artifactory server itself (if the filestore is local, then *cache-fs* is meaningless). In the case of an HA configuration, the *cache-fs* will be mounted and the recommendation is for each node to have its own *cache-fs* layer.

cache-fs template configuration

If you choose to use the *cache-fs* template, your *binarystore.xml* configuration file should look like this:

```
<config version="v1">
  <chain template="cache-fs"/>
</config>
```


What's in the template?

While you don't need to configure anything else in your *binarystore.xml*, this is what the *cache-fs* template looks like under the hood.

This example sets the *cache-fs* size to be 10GB and its location (absolute path since it starts with a "/") to be */cache/filestore*.

```
<config version="v1">
  <chain template="cache-fs" />
  <provider id="cache-fs" type="cache-fs">
    <cacheProviderDir>/cache/filestore</cacheProviderDir>
    <maxCacheSize>10000000000</maxCacheSize>
  </provider>
</config>
```

Where:

type	cache-fs
maxCacheSize	Default: 5000000000 (5GB) The maximum storage allocated for the cache in bytes . Please note that <i>maxCacheSize</i> does not include files that are in progress of being uploaded (which is saved under <i>cache/_pre</i>); thus it is recommended to keep extra spaces for <i>_pre</i> folder.
cacheProviderDir	Default: cache The root folder of binaries for the filestore cache. If the value specified starts with a forward slash ("/") it is considered the fully qualified path to the filestore folder. Otherwise, it is considered relative to the baseDataDir .

Full-DB Binary Provider

This binary provider saves all the metadata and binary content as BLOBs in the database with an additional layer of caching on the filesystem.

Caching can improve Artifactory's performance since frequent requests will be served from the *cache-fs* before reaching out to the database.

full-db template configuration

If you choose to use the *full-db* template, your *binarystore.xml* configuration file should look like this:

```
<config version="v1">
  <chain template="full-db" />
</config>
```

What's in the template?

While you don't need to configure anything else in your *binarystore.xml*, this is what the *full-db* template looks like under the hood.

For details about the *cache-fs* provider, please refer to [Cached Filesystem Binary Provider](#).

The *blob* provider is what handles the actual saving of metadata and binary content as BLOBs in the database.

```
<config version="v1">
  <chain template="full-db" />
  <provider id="cache-fs" type="cache-fs">
    <provider id="blob" type="blob" />
  </provider>
</config>
```

Full-DB-Direct Binary Provider

This binary provider saves all the metadata and binary content as BLOBs in the database without using a caching layer.

full-db-direct template configuration

If you choose to use the *full-db-direct* template, your *binarystore.xml* configuration file should look like this:

```
<config version="v1">
  <chain template="full-db-direct" />
</config>
```

What's in the template?

While you don't need to configure anything else in your *binarystore.xml*, this is what the *full-db-direct* template looks like under the hood. The *blob* provider is what handles the actual saving of metadata and binary content as BLOBs in the database.

```
<config version="v1">
  <chain template="full-db-direct" />
  <provider id="blob" type="blob" />
</config>
```

Cloud Storage Providers

Using cloud storage providers is only available with an Enterprise license.



As part of its universal approach, Artifactory supports a variety of cloud storage providers described in detail in the sections below. These providers will typically be wrapped with other binary providers to ensure that the binary resources are always available from Artifactory (for example, to enable Artifactory to serve files when requested even if they have not yet reached the cloud storage due to upload latency).

S3 Binary Provider

Artifactory provides templates to let you configure storage on an S3 cloud provider where there are two options: s3 and s3old

- The s3 template is used for configuring S3 Object Storage using the JetS3t library.
- The s3Old template is used for configuring S3 Object Storage using the JClouds.

These binary providers for cloud storage solutions have a very similar selection of parameters.

type	s3 or s3old
testConnection	Default: true When set to true, the binary provider uploads and downloads a file when Artifactory starts up to verify that the connection to the cloud storage provider is fully functional.
useSignature	Default: false. When set to true, requests to AWS S3 are signed. Available from AWS S3 version 4. For details, please refer to Signing AWS API requests in the AWS S3 documentation.
multiPartLimit	Default: 100,000,000 bytes File size threshold over which file uploads are chunked and multi-threaded.

identity	Your cloud storage provider identity.
credential	Your cloud storage provider authentication credential.
region	The region offered by your cloud storage provider with which you want to work.
bucketName	Your globally unique bucket name.
path	Default: <code>filestore</code> The path relative to the bucket where binary files are stored.
rootFoldersNameLength	Default: 2 The number of initial characters in the object's checksum that should be used to name the folder in storage. This can take any value between 0 - 5. 0 means that checksum files will be stored at the root of the object store bucket. For example, if the object's checksum is <code>8c335149...</code> and <code>rootFoldersNameLength</code> is set to 4, the folder under which the object would be stored would be named <code>8c33</code> .
proxyIdentity	Corresponding parameters if you are accessing the cloud storage provider through a proxy server.
proxyCredential	
proxyPort	
proxyHost	
port	The cloud storage provider's port.
endPoint	The cloud storage provider's URL.
roleName	Only available on S3 . The IAM role configured on your Amazon server for authentication. When this parameter is used, the refreshCredentials parameter must be set to true.
refreshCredentials	Default: false. Only available on S3 . When true, the owner's credentials are automatically renewed if they expire. When roleName is used, this parameter must be set to true.
httpsOnly	Default: true. Only available on S3 . Set to true if you only want to access your cloud storage provider through a secure https connection.

httpsPort	<p>Default: 443. Must be set if httpsOnly is true. The https port for the secure connection.</p> <p>When this value is specified, the port needs to be removed from the endPoint.</p>
providerID	Set to S3. Only available for S3old .
s3AwsVersion	<p>Default: 'AWS4-HMAC-SHA256' (AWS signature version 4). Only available on S3.</p> <p>Can be set to 'AWS2' if AWS signature version 2 is needed. Please refer the AWS documentation for more information.</p>
<property name="s3service.disable-dns-buckets" value="true"></property>	<p>Artifactory by default prepends the bucketName in front of the endpoint (e.g. mybucket.s3.aws.com) to create an URL that it access the S3 bucket with. S3 providers such as Amazon AWS uses this convention.</p> <p>However, this is not the case for some S3 providers use the bucket name as part of the context URL (e.g. s3provider.com/mybucket); so Artifactory needs to have following perimeter added in order for the URI to be compatible with the S3 providers. S3 providers that use this URI format includes OpenStack, CEPH, CleverSafe, and EMC ECS.</p>

The snippets below show the basic template configuration and examples that use the S3 binary provider to support several configurations (CEPH, CleverSafe and more).

s3 template configuration

Because you must configure the s3 provider with parameters specific to your account (but can leave all other parameters with the recommended values), if you choose to use this template, your *binarystore.xml* configuration file should look like this:

```
<config version="2">
  <chain template="s3"/>
  <provider id="s3" type="s3">
    <endpoint>http://s3.amazonaws.com</endpoint>
    <identity>[ENTER IDENTITY HERE]</identity>
    <credential>[ENTER CREDENTIALS HERE]</credential>
    <path>[ENTER PATH HERE]</path>
    <bucketName>[ENTER BUCKET NAME HERE]</bucketName>
  </provider>
</config>
```

What's in the template?

While you don't need to configure anything else in your *binarystore.xml*, this is what the s3 template looks like under the hood.

```

<config version="v1">
  <chain template="s3"/>
    <provider id="cache-fs" type="cache-fs">
      <provider id="eventual" type="eventual">
        <provider id="retry" type="retry">
          <provider id="s3" type="s3"/>
        </provider>
      </provider>
    </provider>
  </provider>
</config>

```

For details about the *cache-fs* provider, please refer to [Cached Filesystem Binary Provider](#).

For details about the *eventual* provider, please refer to [Eventual Binary Provider](#).

For details about the *retry* provider, please refer to [Retry Binary Provider](#).

Example 1

A configuration for OpenStack Object Store Swift.

```

<config version="v1">
  <chain template="s3"/>
    <provider id="s3" type="s3">
      <identity>XXXXXXXXXX</identity>
      <credential>XXXXXXXXXX</credential>
      <endpoint><My OpenStack Server></endpoint>
      <bucketName><My OpenStack Container></bucketName>
      <httpsOnly>false</httpsOnly>
      <property name="s3service.disable-dns-buckets"
value="true"></property>
    </provider>
  </config>

```

Example 2

A configuration for CEPH.

```

<config version="v1">
  <chain template="s3"/>
  <provider id="s3" type="s3">
    <identity>XXXXXXXXXX</identity>
    <credential>XXXXXXXXXXXXXXXXXX</credential>
    <endpoint><My Ceph server></endpoint>      <!-- Specifies the CEPH
endpoint -->
    <bucketName>[My Ceph Bucket Name]</bucketName>
    <property name="s3service.disable-dns-buckets"
value="true"></property>
    <httpsOnly>>false</httpsOnly>
  </provider>
</config>

```

Example 3

A configuration for CleverSafe.

```

<config version="v1">
  <chain template="s3"/>
  <provider id="s3" type="s3">
    <identity>XXXXXXXXXX</identity>
    <credential>XXXXXXXXXX</credential>
    <endpoint>[My CleverSafe Server]</endpoint>  <!-- Specifies the
CleverSafe endpoint -->
    <bucketName>[My CleverSafe Bucket]</bucketName>
    <httpsOnly>>false</httpsOnly>
    <property name="s3service.disable-dns-buckets"
value="true"></property>
  </provider>
</config>

```

Example 4

A configuration for S3 with a proxy between Artifactory and the S3 bucket.

```

<config version="v1">
  <chain template="s3"/>
  <provider id="s3" type="s3">
    <identity>XXXXXXXXXX</identity>
    <credential>XXXXXXXXXXXXXXXXXX</credential>
    <endpoint>[My S3 server]</endpoint>
    <bucketName>[My S3 Bucket Name]</bucketName>
    <proxyHost>[http proxy host name]</proxyHost>
    <proxyPort>[http proxy port number]</proxyPort>
    <proxyIdentity>XXXXX</proxyIdentity>
    <proxyCredential>XXXX</proxyCredential>
  </provider>
</config>

```

Example 5

A configuration for S3 using an IAM role instead of an IAM user.

```

<config version="v1">
  <chain template="s3"/>
  <provider id="s3" type="s3">
    <roleName>XXXXXX</roleName>
    <endpoint>s3.amazonaws.com</endpoint>
    <bucketName>[mybucketname]</bucketName>
    <refreshCredentials>true</refreshCredentials>
  </provider>
</config>

```

Example 6

A configuration for S3 when using server side encryption.

```

<config version="v1">
  <chain template="s3"/>
  <provider id="s3" type="s3">
    <identity>XXXXXXXXXX</identity>
    <credential>XXXXXXXXXX</credential>
    <endpoint>s3.amazonaws.com</endpoint>
    <bucketName>[mybucketname]</bucketName>
    <property name="s3service.server-side-encryption"
value="AES256"></property>
  </provider>
</config>

```

Example 7

A configuration for S3 when using EMC Elastic Cloud Storage (ECS).

```

<config version="v1">
  <chain template="s3"/>
    <provider id="s3" type="s3">
      <identity>XXXXXXXXXX</identity>
      <credential>XXXXXXXXXXXXXXXXXXXX</credential>
      <endpoint><My ECS server></endpoint>      <!-- e.g.
https://emc-ecs.mycompany.com -->
      <httpsPort><My ECS Server SSL Port></httpsPort>      <!--
Required only if HTTPS port other than 443 is used -->
      <bucketName>[My ECS Bucket Name]</bucketName>
      <property name="s3service.disable-dns-buckets"
value="true"></property>
    </provider>
  </config>

```

S3Old Binary Provider

The snippet below shows an example that uses the S3 binary provider where JClouds is the underlying framework.

s3Old template configuration

A configuration for AWS.

Because you must configure the *s3Old* provider with parameters specific to your account (but can leave all other parameters with the recommended values), if you choose to use this template, your *binarystore.xml* configuration file should look like this:

```

<config version="v1">
  <chain template="s3Old"/>
    <provider id="s3Old" type="s3Old">
      <identity>XXXXXXXXXX</identity>
      <credential>XXXXXXXXXX</credential>
      <endpoint>s3.amazonaws.com</endpoint>
      <bucketName>[mybucketname]</bucketName>
    </provider>
  </config>

```

What's in the template?

While you don't need to configure anything else in your *binarystore.xml*, this is what the *s3Old* template looks like under the hood.


```

<config version="v1">
  <chain template="s3Old"/>
  <provider id="cache-fs" type="cache-fs">
    <provider id="eventual" type="eventual">
      <provider id="retry" type="retry">
        <provider id="s3Old" type="s3Old"/>
      </provider>
    </provider>
  </provider>
</config>

```

For details about the *cache-fs* provider, please refer to [Cached Filesystem Binary Provider](#).

For details about the *eventual* provider, please refer to [Eventual Binary Provider](#).

For details about the *retry* provider, please refer to [Retry Binary Provider](#).

Google Storage Binary Provider

The google-storage template is used for configuring Google Cloud Storage as the remote filestore.

The snippets below show the basic template configuration and an examples that use the Google Cloud Storage binary provider.

This binary provider uses the following set of parameters:

type	google-storage
testConnection	Default: true When set to true, the binary provider uploads and downloads a file when Artifactory starts up to verify that the connection to the cloud storage provider is fully functional.
multiPartLimit	Default: 100,000,000 bytes File size threshold over which file uploads are chunked and multi-threaded.
identity	Your cloud storage provider identity.
credential	Your cloud storage provider authentication credential.
bucketName	Your globally unique bucket name.
path	Default: filestore The path relative to the bucket where binary files are stored.
proxyIdentity	Corresponding parameters if you are accessing the cloud storage provider through a proxy server.
proxyCredential	
proxyPort	
proxyHost	

port	The cloud storage provider's port.
endPoint	The cloud storage provider's URL.
httpsOnly	Default: true. Set to true if you only want to access your cloud storage provider through a secure https connection.
httpsPort	Default: 443. Must be set if httpsOnly is true . The https port for the secure connection. When this value is specified, the port needs to be removed from the endPoint .
bucketExists	Default: false. When set to true, it indicates to the binary provider that a bucket already exists in Google Cloud Storage and therefore does not need to be created.

google-storage template configuration

Because you must configure the *google-storage* provider with parameters specific to your account (but can leave all other parameters with the recommended values), if you choose to use this template, your *binarystore.xml* configuration file should look like this:

```
<config version="v1">
  <chain template="google-storage"/>

  <provider id="google-storage" type="google-storage">
    <endpoint>comondatastorage.googleapis.com</endpoint>
    <bucketName><BUCKET NAME></bucketName>
    <identity>XXXXXXX</identity>
    <credential>XXXXXXX</credential>
  </provider>
</config>
```

What's in the template?

While you don't need to configure anything else in your *binarystore.xml*, this is what the *google-storage* template looks like under the hood.

```
<config version="v1">
  <chain template="google-storage"/>
    <provider id="cache-fs" type="cache-fs">
      <provider id="eventual" type="eventual">
        <provider id="retry" type="retry">
          <provider id="google-storage" type="google-storage"/>
        </provider>
      </provider>
    </provider>
  </provider>
</config>
```

For details about the *cache-fs* provider, please refer to [Cached Filesystem Binary Provider](#).

For details about the *eventual* provider, please refer to [Eventual Binary Provider](#).

For details about the *retry* provider, please refer to [Retry Binary Provider](#).

Example 1

A configuration with a dynamic property from the JetS3t library. In this example, the `httpClient.max-connections` parameter sets the maximum number of simultaneous connections to allow globally (default is 100).

```
<config version="v1">
  <chain template="google-storage"/>
  <provider id="google-storage" type="google-storage">
    <endpoint>commondatastorage.googleapis.com</endpoint>
    <bucketName><BUCKET NAME></bucketName>
    <identity>XXXXXX</identity>
    <credential>XXXXXX</credential>
    <property name="httpClient.max-connections" value=150></property>
  </provider>
</config>
```

Azure Blob Storage Binary Provider

The `azure-blob-storage` template is used for configuring Azure Blob Storage as the remote filestore.

The snippets below show the basic template configuration and an examples that use the Azure Blob Storage binary provider.

This binary provider uses the following set of parameters:

testConnection	Default: true When true, Artifactory uploads and downloads a file when starting up to verify that the connection to the cloud storage provider is fully functional.
accountName	The storage account can be a General-purpose storage account or a Blob storage account which is specialized for storing objects/blobs. Your cloud storage provider identity.
accountKey	Your cloud storage provider authentication credential.
containerName	Your globally unique container name on Azure Blob Storage.
endpoint	The hostname. You should only use the default value unless you need to contact a different endpoint for testing purposes.
httpsOnly	Default: true. Set to true if you only want to access through a secure https connection.

The following snippet shows the default chain that uses `azure-blob-storage` as the binary provider:

```

<config version="1">
  <chain template="azure-blob-storage"/>
  <provider id="azure-blob-storage" type="azure-blob-storage">
    <accountName>XXXXXXXXX</accountName>
    <accountKey>XXXXXXXXX</accountKey>

<endpoint>https://<ACCOUNT_NAME>.blob.core.windows.net/</endpoint>
  <containerName><NAME></containerName>
  </provider>
</config>

```

Eventual Binary Provider

This binary provider is not independent and will always be used as part of a template chain for a remote filestore that may exhibit upload latency (e.g. S3 or GCS). To overcome potential latency, files are first written to a folder called "eventual" under the **baseDataDir** in local storage, and then later uploaded to persistent storage with the cloud provider. The default location of the `eventual` folder is under the `$ARTIFACTORY_HOME/data` folder (or `$CLUSTER_HOME/ha-data` in the case of an HA configuration using a version of Artifactory **below 5.0**) and is not configurable. You need to make sure that Artifactory has full read/write permissions to this location.

There are three additional folders under the `eventual` folder:

- `_pre`: part of the persistence mechanism that ensures all files are valid before being uploaded to the remote filestore
- `_add`: handles upload of files to the remote filestore
- `_delete`: handles deletion of files from the remote filestore

Example

The example below shows a configuration that uses S3 for persistent storage after temporary storage with an eventual binary provider. The eventual provider configures 10 parallel threads for uploading and a lock timeout of 180 seconds.

```

<!-- The S3 binary provider configuration -->
<config version="v1">
  <chain template="s3"/>
  <provider id="s3" type="s3">
    <identity>XXXXXXXXXX</identity>
    <credential>XXXXXXXXXX</credential>
    <endpoint><My OpenStack Server></endpoint>
    <bucketName><My OpenStack Container></bucketName>
    <httpsOnly>false</httpsOnly>
    <property name="s3service.disable-dns-buckets"
value="true"></property>
  </provider>

<!-- The eventual provider configuration -->
  <provider id="eventual" type="eventual">
    <numberOfThreads>10</numberOfThreads>
    <timeout>180000</timeout>
  </provider>
</config>

```

Where:

type	eventual
-------------	----------

timeout	The maximum amount of time a file may be locked while it is being written to or deleted from the filesystem.
dispatchInterval	Default: 5000 ms The interval between which the provider scans the "eventual" folder to check for files that should be uploaded to persistent storage.
numberOfThreads	Default: 5 The number of parallel threads that should be allocated for uploading files to persistent storage.

Retry Binary Provider

This binary provider is not independent and will always be used as part of a more complex template chain of providers. In case of a failure in a read or write operation, this binary provider notifies its underlying provider in the hierarchy to retry the operation.

type	retry
interval	Default: 5000 ms The time interval to wait before retries.
maxTrys	Default: 5 The maximum number of attempts to read or write before responding with failure.

Example

The example below shows a configuration that uses S3 for persistent storage , but uses a retry provider to keep retrying (up to a maximum of 10 times) in case upload fails.

```

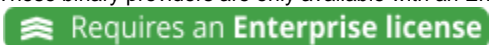
<!-- The S3 binary provider configuration -->
<config version="v1">
  <chain template="s3"/>
  <provider id="s3" type="s3">
    <identity>XXXXXXXXXX</identity>
    <credential>XXXXXXXXXX</credential>
    <endpoint><My OpenStack Server></endpoint>
    <bucketName><My OpenStack Container></bucketName>
    <httpsOnly>false</httpsOnly>
    <property name="s3service.disable-dns-buckets"
value="true"></property>
  </provider>

  <!-- The retry provider configuration -->
  <provider id="retry" type="retry">
    <maxTrys>10</maxTrys>
  </provider>
</config>

```

Double Shards, Redundant Shards

These binary providers are only available with an Enterprise license.



Double Shards Binary Provider

The *double-shards* template is used for pure sharding configuration that uses 2 physical mounts with 1 copy (which means each artifact is saved only once). To learn more about the different sharding capabilities, refer to [Filestore Sharding](#).

double-shards template configuration

If you choose to use the *double-shards* template, your *binarystore.xml* configuration file should look like this:

```
<config version="4">
  <chain template="double-shards" />
</config>
```

What's in the template?

While you don't need to configure anything else in your *binarystore.xml*, this is what the *double-shards* template looks like under the hood. For details about the *cache-fs* provider, please refer to [Cached Filesystem Binary Provider](#). For details about the *sharding* provider, please refer to [Sharding Binary Provider](#). For details about the *state-aware* sub-provider, please refer to [State-Aware Binary Provider](#).

```
<config version="4">
  <chain template="double-shards" />
  <provider id="cache-fs" type="cache-fs">
    <provider id="sharding" type="sharding">
      <redundancy>1</redundancy>
      <sub-provider id="shard-fs-1" type="state-aware" />
      <sub-provider id="shard-fs-2" type="state-aware" />
    </provider>
  </provider>
</config>
```

Redundant Shards Binary Provider

The *redundant-shards* template is used for pure sharding configuration that uses 2 physical mounts with 2 copies (which means each shard stores a copy of each artifact). To learn more about the different sharding capabilities, refer to [Filestore Sharding](#).

redundant-shards template configuration

If you choose to use the *redundant-shards* template, your *binarystore.xml* configuration file should look like this:

```
<config version="4">
  <chain template="redundant-shards" />
</config>
```

What's in the template?

While you don't need to configure anything else in your *binarystore.xml*, this is what the *redundant-shards* template looks like under the hood. Details about the *cache-fs* provider can be found in the [Cached Filesystem Binary Provider](#) section. Details about the *sharding* provider can be found in the [Sharding Binary Provider](#) section. Details about the *state-aware* sub-provider can be found in the [State-Aware Binary Provider](#) section.

```

<config version="4">
  <chain template="redundant-shards"/>
  <provider id="cache-fs" type="cache-fs">
    <provider id="sharding" type="sharding">
      <redundancy>2</redundancy>
      <sub-provider id="shard-state-aware-1" type="state-aware"/>
      <sub-provider id="shard-state-aware-2" type="state-aware"/>
    </provider>
  </provider>
</config>

```

Sharding Binary Provider

Artifactory offers a Sharding Binary Provider that lets you manage your binaries in a sharded filestore. A sharded filestore is one that is implemented on a number of physical mounts (M), which store binary objects with redundancy (R), where $R \leq M$. This binary provider is not independent and will always be used as part of a more complex template chain of providers. To learn about sharding, refer to [Filestore Sharding](#).

type	sharding
readBehavior	<p>This parameter dictates the strategy for reading binaries from the mounts that make up the sharded filestore.</p> <p>Possible values are:</p> <p>roundRobin (default): Binaries are read from each mount using a round robin strategy.</p>
writeBehavior	<p>This parameter dictates the strategy for writing binaries to the mounts that make up the sharded filestore.</p> <p>Possible values are:</p> <p>roundRobin (default): Binaries are written to each mount using a round robin strategy.</p> <p>freeSpace: Binaries are written to the mount with the greatest absolute volume of free space available.</p> <p>percentageFreeSpace: Binaries are written to the mount with the percentage of free space available.</p>
redundancy	<p>Default: $r = 1$</p> <p>The number of copies that should be stored for each binary in the filestore. Note that redundancy must be less than or equal to the number of mounts in your system for Artifactory to work with this configuration.</p>
concurrentStreamWaitTimeout	<p>Default: 30,000 ms</p> <p>To support the specified redundancy, accumulates the write stream in a buffer, and uses "r" threads (according to the specified redundancy) to write to each of the redundant copies of the binary being written. A binary can only be considered written once all redundant threads have completed their write operation. Since all threads are competing for the write stream buffer, each one will complete the write operation at a different time. This parameter specifies the amount of time (ms) that any thread will wait for all the others to complete their write operation.</p> <div style="border: 1px solid green; padding: 5px; margin-top: 10px;"> <p>If a write operation fails, you can try increasing the value of this parameter.</p> </div>
concurrentStreamBufferKb	<p>Default: 32 Kb</p> <p>The size of the write buffer used to accumulate the write stream before being replicated for writing to the "r" redundant copies of the binary.</p> <div style="border: 1px solid green; padding: 5px; margin-top: 10px;"> <p>If a write operation fails, you can try increasing the value of this parameter.</p> </div>

maxBalancingRunTime	<p>Default: 3,600,000 ms (1 hour)</p> <p>Once a failed mount has been restored, this parameter specifies how long each balancing session may run before it lapses until the next Garbage Collection has completed. For more details about balancing, please refer to Using Balancing to Recover from Mount Failure.</p> <div style="border: 1px solid green; padding: 5px; margin-top: 10px;"> <p>To restore your system to full redundancy more quickly after a mount failure, you may increase the value of this parameter. If you find this causes an unacceptable degradation of overall system performance, you can consider decreasing the value of this parameter, but this means that the overall time taken for Artifactory to restore full redundancy will be longer.</p> </div>
freeSpaceSampleInterval	<p>Default: 3,600,000 ms (1 hour)</p> <p>To implement its write behavior, Artifactory needs to periodically query the mounts in the sharded filestore to check for free space. Since this check may be a resource intensive operation, you may use this parameter to control the time interval between free space checks.</p> <div style="border: 1px solid green; padding: 5px; margin-top: 10px;"> <p>If you anticipate a period of intensive upload of large volumes of binaries, you can consider decreasing the value of this parameter in order to reduce the transient imbalance between mounts in your system.</p> </div>
minSpareUploaderExecutor	<p>Default: 2</p> <p>Artifactory maintains a pool of threads to execute writes to each redundant unit of storage. Depending on the intensity of write activity, eventually, some of the threads may become idle and are then candidates for being killed. However, Artifactory does need to maintain some threads alive for when write activities begin again. This parameter specifies the minimum number of threads that should be kept alive to supply redundant storage units.</p>
uploaderCleanupIdleTime	<p>Default: 120,000 ms (2 min)</p> <p>The maximum period of time threads may remain idle before becoming candidates for being killed.</p>

State-Aware Binary Provider

This binary provider is not independent and will always be used in the [sharding](#) or [sharding-cluster](#) providers. The provider is aware if its underlying disk is functioning or not. It is identical to the basic [filesystem](#) provider provider, however, it can also recover from errors (the parent provider is responsible for recovery) with the addition of the `checkPeriod` field.

type	state-aware
checkPeriod	<p>Default: 15000 ms</p> <p>The minimum time to wait between trying to re-activate the provider if it had fatal errors at any point.</p>
zone	The name of the sharding zone the provider is part of (only applicable under a sharding provider)

Configuring Sharding for HA Cluster

These binary providers are only available with an Enterprise license.



For a High Availability cluster, Artifactory offers templates that support sharding-cluster for File-System, S3 and Google Storage. To learn more about the different sharding capabilities, refer to [Filestore Sharding](#).

When configuring your filestore on an HA cluster, you need to place the `binarystore.xml` under `$ARTIFACTORY_HOME/etc` in the primary node and it will be synced to the other members in the cluster.

File System Cluster Binary Provider

When using the *cluster-file-system* template, each node has its own local filestore (just like in the [file-system binary provider](#)) and is connected to all other cluster nodes via dynamically allocated Remote Binary Providers using the [Sharding-Cluster Binary Provider](#).

cluster-file-system template configuration

If you choose to use the *cluster-file-system* template, your *binarystore.xml* configuration file should look like this:

```
<config version="2">
  <chain template="cluster-file-system" />
</config>
```

What's in the template?

While you don't need to configure anything else in your *binarystore.xml*, this is what the *cluster-file-system* template looks like under the hood.

Details about the *cache-fs* provider can be found in the [Cached Filesystem Binary Provider](#) section.

Details about the *sharding-cluster* can be found in the [Sharding-Cluster Binary Provider](#) section.

Details about the *state-aware* sub-provider can be found in the [State-Aware Binary Provider](#) section.

```
<config version="2">
  <chain> <!--template="cluster-file-system"-->
    <provider id="cache-fs" type="cache-fs">
      <provider id="sharding-cluster" type="sharding-cluster">
        <sub-provider id="state-aware" type="state-aware"/>
        <dynamic-provider id="remote-fs" type="remote"/>
      </provider>
    </provider>
  </chain>

  <provider id="state-aware" type="state-aware">
    <zone>local</zone>
  </provider>

  <!-- Shard dynamic remote provider configuration -->
  <provider id="remote-fs" type="remote">
    <zone>remote</zone>
  </provider>

  <provider id="sharding-cluster" type="sharding-cluster">
    <readBehavior>crossNetworkStrategy</readBehavior>
    <writeBehavior>crossNetworkStrategy</writeBehavior>
    <redundancy>2</redundancy>
    <property name="zones" value="local,remote"/>
  </provider>

</config>
```

S3 Cluster Binary Provider

This is the setting used for [S3 Object Storage](#) using the JetS3t library when configuring filestore sharding for an HA cluster. It is based on the sharding and dynamic provider logic that synchronizes the *cluster-file-system*.

When using the *cluster-s3* template, data is temporarily stored on the file system of each node using the [Eventual Binary Provider](#), and is then passed on to your S3 object storage for persistent storage.

Each node has its own local filestore (just like in the [file-system binary provider](#)) and is connected to all other cluster nodes via dynamically allocated Remote Binary Providers using the [Sharding-Cluster Binary Provider](#).

cluster-s3 template configuration

Because you must configure the s3 provider with parameters specific to your account (but can leave all other parameters with the recommended values), if you choose to use the *cluster-s3* template, your *binarystore.xml* configuration file should look like this:

```
<config version="2">
  <chain template="cluster-s3"/>
  <provider id="s3" type="s3">
    <endpoint>http://s3.amazonaws.com</endpoint>
    <identity>[ENTER IDENTITY HERE]</identity>
    <credential>[ENTER CREDENTIALS HERE]</credential>
    <path>[ENTER PATH HERE]</path>
    <bucketName>[ENTER BUCKET NAME HERE]</bucketName>
  </provider>
</config>
```

What's in the template?

While you don't need to configure anything else in your *binarystore.xml*, this is what the *cluster-s3* template looks like under the hood.

```

<config version="2">
  <chain> <!--template="cluster-s3"-->
    <provider id="cache-fs-eventual-s3" type="cache-fs">
      <provider id="sharding-cluster-eventual-s3"
type="sharding-cluster">
        <sub-provider id="eventual-cluster-s3"
type="eventual-cluster">
          <provider id="retry-s3" type="retry">
            <provider id="s3" type="s3"/>
          </provider>
        </sub-provider>
        <dynamic-provider id="remote-s3" type="remote"/>
      </provider>
    </provider>
  </chain>

  <provider id="sharding-cluster-eventual-s3" type="sharding-cluster">
    <readBehavior>crossNetworkStrategy</readBehavior>
    <writeBehavior>crossNetworkStrategy</writeBehavior>
    <redundancy>2</redundancy>
    <property name="zones" value="local,remote"/>
  </provider>

  <provider id="remote-s3" type="remote">
    <zone>remote</zone>
  </provider>

  <provider id="eventual-cluster-s3" type="eventual-cluster">
    <zone>local</zone>
  </provider>
  <provider id="s3" type="s3">
    <endpoint>http://s3.amazonaws.com</endpoint>
    <identity>[ENTER IDENTITY HERE]</identity>
    <credential>[ENTER CREDENTIALS HERE]</credential>
    <path>[ENTER PATH HERE]</path>
    <bucketName>[ENTER BUCKET NAME HERE]</bucketName>
  </provider>
</config>

```

Details about the *cache-fs* provider can be found in the [Cached Filesystem Binary Provider](#) section.

Details about the *sharding-cluster* can be found in the [Sharding-Cluster Binary Provider](#) section.

Details about the *eventual-cluster* sub-provider can be found in the [Eventual Binary Provider](#) section.

Details about the *retry* provider can be found in the [Retry Binary Provider](#) section.

Details about the *remote* dynamic provider can be found in the [Remote Binary Provider](#) section.

Google Storage Cluster Binary Provider

This is the setting used for [Google Cloud Storage](#) using the JetS3t library when configuring filestore sharding for an HA cluster. It is based on the sharding and dynamic provider logic that synchronizes the cluster-file-system.

When using the *cluster-google-storage* template, data is temporarily stored on the file system of each node using the [Eventual Binary Provider](#), and is then passed on to your Google storage for persistent storage.

Each node has its own local filestore (just like in the [file-system binary provider](#)) and is connected to all other cluster nodes via dynamically allocated [Remote Binary Providers](#) using the [Sharding-Cluster Binary Provider](#).

cluster-google-storage template configuration

Because you must configure the *google-storage* provider with parameters specific to your account (but can leave all other parameters with the recommended values), if you choose to use the *cluster-google-storage* template, your *binarystore.xml* configuration file should look like this:

```
<config version="2">
  <chain template="cluster-google-storage"/>
  <provider id="google-storage" type="google-storage">
    <endpoint>commondatastorage.googleapis.com</endpoint>
    <bucketName><BUCKET NAME></bucketName>
    <identity>XXXXXXX</identity>
    <credential>XXXXXXXX</credential>
  </provider>
</config>
```

What's in the template?

While you don't need to configure anything else in your *binarystore.xml*, this is what the *cluster-google-storage* template looks like under the hood.

```

<config version="2">
  <chain> <!--template="cluster-google-storage"-->
    <provider id="cache-fs-eventual-google-storage" type="cache-fs">
      <provider id="sharding-cluster-eventual-google-storage"
type="sharding-cluster">
        <sub-provider id="eventual-cluster-google-storage"
type="eventual-cluster">
          <provider id="retry-google-storage" type="retry">
            <provider id="google-storage"
type="google-storage"/>
          </provider>
        </sub-provider>
        <dynamic-provider id="remote-google-storage"
type="remote"/>
      </provider>
    </provider>
  </chain>

  <provider id="sharding-cluster-eventual-google-storage"
type="sharding-cluster">
    <readBehavior>crossNetworkStrategy</readBehavior>
    <writeBehavior>crossNetworkStrategy</writeBehavior>
    <redundancy>2</redundancy>
    <property name="zones" value="local,remote"/>
  </provider>

  <provider id="remote-google-storage" type="remote">
    <zone>remote</zone>
  </provider>

  <provider id="eventual-cluster-google-storage" type="eventual-cluster">
    <zone>local</zone>
  </provider>

  <provider id="google-storage" type="google-storage">
    <endpoint>commondatastorage.googleapis.com</endpoint>
    <bucketName><BUCKET NAME></bucketName>
    <identity>XXXXXX</identity>
    <credential>XXXXXXXX</credential>
  </provider>
</config>

```

Details about the *cache-fs* provider can be found in the [Cached Filesystem Binary Provider](#) section.
 Details about the *sharding-cluster* can be found in the [Sharding-Cluster Binary Provider](#) section.
 Details about the *eventual-cluster* sub-provider can be found in the [Eventual Binary Provider](#) section.
 Details about the *retry* provider can be found in the [Retry Binary Provider](#) section.
 Details about the *remote* dynamic provider can be found in the [Remote Binary Provider](#) section.

Azure Blob Storage Cluster Binary Provider

This is the setting used for **Azure Blob Storage**. It is based on the sharding and dynamic provider logic that synchronizes the cluster-file-system.
 When using the *cluster-azure-blob-storage* template, data is temporarily stored on the file system of each node using the [Eventual Binary Provider](#), and is then passed on to your Azure Blob Storage for persistent storage.

Each node has its own local filestore (just like in the [file-system binary provider](#)) and is connected to all other cluster nodes via dynamically allocated Remote Binary Providers using the Sharding-Cluster Binary Provider.

cluster-azure-blob-storage template configuration

Because you must configure the *azure-blob-storage* provider with parameters specific to your account (but can leave all other parameters with the recommended values), if you choose to use the *cluster-azure-blob-storage* template, your *binarystore.xml* configuration file should look like this:

```
<config version="2">
  <chain template="cluster-azure-blob-storage"/>
    <provider id="azure-blob-storage" type="azure-blob-storage">
      <accountName>XXXXXXX</accountName>
      <accountKey>XXXXXXX</accountKey>

      <endpoint>https://<ACCOUNT_NAME>.blob.core.windows.net/</endpoint>
      <containerName><NAME></containerName>
    </provider>
</config>
```

What's in the template?

While you don't need to configure anything else in your *binarystore.xml*, this is what the *cluster-azure-blob-storage* template looks like under the hood:

```

<config version="2">
  <chain template="cluster-azure-blob-storage">
    <provider id="cache-fs-eventual-azure-blob-storage"
type="cache-fs">
      <provider id="sharding-cluster-eventual-azure-blob-storage"
type="sharding-cluster">
        <sub-provider id="eventual-cluster-azure-blob-storage"
type="eventual-cluster">
          <provider id="retry-azure-blob-storage" type="retry">
            <provider id="azure-blob-storage"
type="azure-blob-storage"/>
          </provider>
        </sub-provider>
        <dynamic-provider id="remote-azure-blob-storage"
type="remote"/>
      </provider>
    </provider>
  </chain>

  <!-- cluster eventual Azure Blob Storage Service default chain -->
  <provider id="sharding-cluster-eventual-azure-blob-storage"
type="sharding-cluster">
    <readBehavior>crossNetworkStrategy</readBehavior>
    <writeBehavior>crossNetworkStrategy</writeBehavior>
    <redundancy>2</redundancy>
    <lenientLimit>1</lenientLimit>
    <property name="zones" value="local,remote"/>
  </provider>

  <provider id="remote-azure-blob-storage" type="remote">
    <zone>remote</zone>
  </provider>

  <provider id="eventual-cluster-azure-blob-storage"
type="eventual-cluster">
    <zone>local</zone>
  </provider>

  <!--cluster eventual template-->
  <provider id="azure-blob-storage" type="azure-blob-storage">
    <accountName>XXXXXXX</accountName>
    <accountKey>XXXXXXX</accountKey>
    <endpoint>https://<ACCOUNT_NAME>.blob.core.windows.net/</endpoint>
    <containerName><NAME></containerName>
  </provider>
</config>

```

Details about the *cache-fs* provider can be found in the [Cached Filesystem Binary Provider](#) section.
 Details about the *sharding-cluster* can be found in the [Sharding-Cluster Binary Provider](#) section.
 Details about the *eventual-cluster* sub-provider can be found in the [Eventual Binary Provider](#) section.
 Details about the *retry* provider can be found in the [Retry Binary Provider](#) section.
 Details about the *remote* dynamic provider can be found in the [Remote Binary Provider](#) section.

Sharding-Cluster Binary Provider

The sharding-cluster binary provider can be used together with other binary providers for both local or cloud-native storage. It adds a **crossNetworkStrategy** parameter to be used as read and write behaviors for validation of the redundancy values and the balance mechanism. It must include a [Remote Binary Provider](#) in its dynamic-provider setting to allow synchronizing providers across the cluster.

The Sharding-Cluster provider listens to cluster topology events and creates or removes dynamic providers based on the current state of nodes in the cluster.

type	sharding-cluster
zones	The zones defined in the sharding mechanism. Read/write strategies take providers based on zones.
lenientLimit	<p>Default: 1 (From version 5.4. Note that for filestores configured with a custom chain and not using the built-in templates, the default value of the lenientLimit parameter is 0 to maintain consistency with previous versions.)</p> <p>The minimum number of filestores that must be active for writes to continue. For example, if lenientLimit is set to 2, my setup includes 4 filestores, and 2 of them go down, writing will continue. If a 3rd filestore goes down, writing will stop.</p> <p>Typically this is used to address transient failures of an individual binary store, with the assumption that the balance mechanism will make up for it over time.</p>
dynamic-provider	The type of provider that can be added and removed dynamically based on cluster topology changes. Currently only the Remote Binary Provider is supported as a dynamic provider.

Example


```

<config version="v1">
  <chain>
    <provider id="cache-fs" type="cache-fs">
      <provider id="sharding-cluster" type="sharding-cluster">
        <sub-provider id="state-aware" type="state-aware"/>
        <dynamic-provider id="remote" type="remote"/>
        <property name="zones" value="remote"/>
      </provider>
    </provider>
  </chain>

  <provider id="sharding-cluster" type="sharding-cluster">
    <readBehavior>crossNetworkStrategy</readBehavior>
    <writeBehavior>crossNetworkStrategy</writeBehavior>
    <redundancy>2</redundancy>
    <lenientLimit>1</lenientLimit>
  </provider>

  <provider id="state-aware" type="state-aware">
    <fileStoreDir>filestore1</fileStoreDir>
  </provider>

  <provider id="remote" type="remote">
    <checkPeriod>15000</checkPeriod>
    <connectionTimeout>5000</connectionTimeout>
    <socketTimeout>15000</socketTimeout>
    <maxConnections>200</maxConnections>
    <connectionRetry>2</connectionRetry>
    <zone>remote</zone>
  </provider>
</config>

```

Remote Binary Provider

This binary provider is not independent and will always be used as part of a more complex template chain of providers. In case of a failure in a read or write operation, this binary provider notifies its parent provider in the hierarchy.

The remote Binary Provider links a node to all other nodes in the cluster, meaning it enables each node to 'see' the filestore of every other node.

type	remote
connectionTimeout	Default: 5000 ms Time before timing out an outgoing connection.
socketTimeout	Default: 15000 ms Time before timing out an established connection (i.e. no data is sent over the wire).
maxConnections	Default: 200 Maximum outgoing connections from the provider.

connectionRetry	Default: 2 How many times to retry connecting to the remote endpoint.
zone	The name of the sharding zone the provider is part of (only applicable under a sharding provider).
checkPeriod	Default: 15000 ms The minimum time to wait between trying to re-activate the provider if it had fatal errors at any point.

Example

The following is an example how a remote binary provider may be configured. To see how this can be integrated with a complete *binarystore.xml* configuration, please refer to the example under [Sharding-Cluster Binary Provider](#).

```
<provider id="remote" type="remote">
  <checkPeriod>15000</checkPeriod>
  <connectionTimeout>5000</connectionTimeout>
  <socketTimeout>15000</socketTimeout>
  <maxConnections>200</maxConnections>
  <connectionRetry>2</connectionRetry>
  <zone>remote</zone>
</provider>
```

Configuring a Custom Filestore From Scratch

In addition to the built-in filestore chain templates below, you may construct custom chain template to accommodate any filestore structure you need.

Since the different Binary providers in the filestore must be compatible with each other, misconfiguration might lead to data loss. For configuring a custom filestore, please contact [JFrog Support](#).

Configuring the Filestore for Older Artifactory Versions

For versions of Artifactory below 4.6, the filestore used is configured in the *\$ARTIFACTORY_HOME/etc/storage.properties* file as follows

binary.provider.type	<p>filesystem (default) This means that metadata is stored in the database, but binaries are stored in the file system. The default location is under <i>\$ARTIFACTORY_HOME/data/filestore</i> however this can be modified.</p> <p>fullDb All the metadata and the binaries are stored as BLOBs in the database.</p> <p>cachedFS Works the same way as <i>filesystem</i> but also has a binary LRU (Least Recently Used) cache for upload/download requests. Improves performance of instances with high IOPS (I/O Operations) or slow NFS access.</p> <p>S3 This is the setting used for S3 Object Storage.</p>
binary.provider.cache.maxSize	This value specifies the maximum cache size (in bytes) to allocate on the system for caching BLOBs.
binary.provider.filesystem.dir	If <i>binary.provider.type</i> is set to <i>filesystem</i> this value specifies the location of the binaries (default: <i>\$ARTIFACTORY_HOME/data/filestore</i>).

binary.provider.cache.dir	The location of the cache. This should be set to your \$ARTIFACTORY_HOME directory directly (not on the NFS).
----------------------------------	---

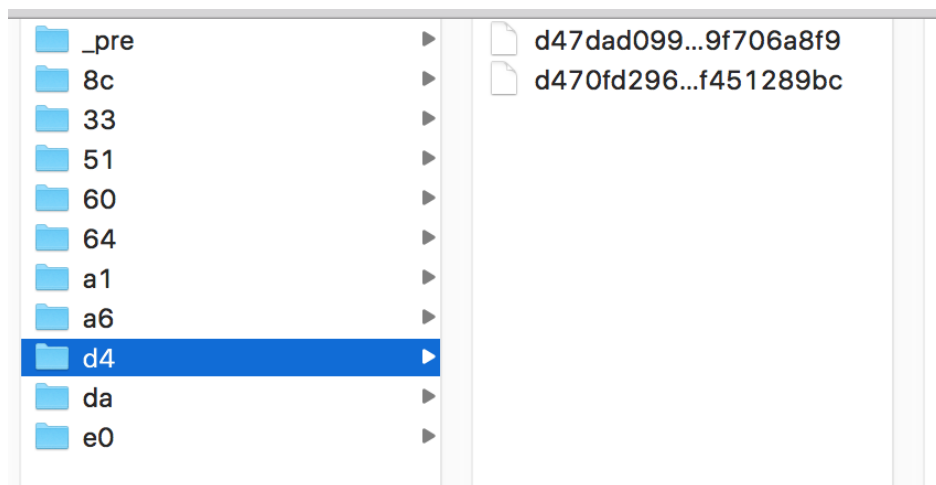
maxCacheSiz

Checksum-Based Storage

Overview

Artifactory uniquely stores artifacts using checksum-based storage.

A file that is uploaded to Artifactory, first has its SHA1 checksum calculated, and is then renamed to its checksum. It is then hosted in the configured filestore in a directory structure made up of the first two characters of the checksum. For example, a file whose checksum is "ac3f5e56..." would be stored in directory "ac"; a file whose checksum is "dfe12a4b..." would be stored in directory "df" and so forth. The example below shows the "d4" directory that contains two files whose checksum begins with "d4".



In parallel, Artifactory's creates a database entry mapping the file's checksum to the path it was uploaded to in a repository. This way of storing binaries optimizes many operations in Artifactory since they are implemented through simple database transactions rather than actually manipulating files.

Deduplication

Artifactory stores any binary file only once. This is what we call "once and once only storage". First time a file is uploaded, Artifactory runs the required checksum calculations when storing the file, however, if the file is uploaded again (to a different location, for example), the upload is implemented as a simple database transaction that creates another record mapping the file's checksum to its new location. There is no need to actually store the file again in storage. No matter how many times a file is uploaded, the filestore only hosts a single copy of the file.

Copying and Moving Files

Copying and moving a file is implemented by simply adding and removing database references and, correspondingly, performance of these actions is that of a database transaction.

Deleting Files

Deleting a file is also a simple database transaction in which the corresponding database record is deleted. The file itself is not directly deleted, even if the last database entry pointing to it is removed. So-called "orphaned" files are removed in the background by Artifactory's garbage collection processes.

Upload, download and replication

Before moving files from one location to another, Artifactory sends checksum headers. If the files already exist in the destination, they are not transferred even if they exist under a different path.

Filesystem Performance

Filesystem performance is greatly improved because actions on the filestore are implemented as database transactions, so there is never any need to do a write-lock on the filesystem.

Page Contents

- Overview
- SHA-256 Support
 - Migrating the Database to Include SHA-256
 - C
 - M

Checksum Search

Searching for a file by its checksum is extremely fast since Artifactory is actually searching through the database for the specified checksum.

Flexible Layout

Since the database is a layer of indirection between the filestore and the displayed layout, any layout can be supported, whether for one of the standard packaging formats such as Maven1, Maven2, npm, NuGet etc. or for any custom layout.

SHA-256 Support

From version 5.5, Artifactory natively supports SHA-256. An artifact's SHA-256 checksum is calculated when it is deployed to Artifactory, and is maintained in persistent storage as part of the database. The [Set Item SHA256 Checksum](#) REST API endpoint (which sets an artifact's SHA-256 checksum as one of its properties) is still supported for backward compatibility, however, this endpoint will eventually be deprecated.

Artifactory's support for SHA-256 checksums is fully-featured and is evident in several ways:

- They can be used in [AQL queries](#), and are returned in corresponding responses
- They are included as download header information
- They can be used in the [Deploy Artifact](#) and [Deploy Artifact by Checksum](#) REST API endpoints.
- They are included when [downloading a folder](#)
- They are displayed in the [General Information](#) tab of the Artifact Repository Browser
- They can be used in a variety of REST API endpoints used for [search](#)

After upgrading to version 5.5 (or above), Artifactory will be fully capable of utilizing an artifact's SHA-256 checksum for any of the features mentioned above.

Making full use of Artifactory's native support for SHA256

New artifacts that are uploaded will automatically have their SHA-256 checksum calculated, however, artifacts that were already hosted in Artifactory prior to the upgrade will not have their SHA-256 checksum in the database yet.

To make full use of Artifactory's SHA-256 capabilities, you need to run a process that [migrates Artifactory's database](#) making sure that the record for each artifact includes its SHA-256 checksum.

Migrating the Database to Include SHA-256

Migrating the database may be a resource intensive operation

Depending on the size of your database, this process may be resource intensive. To mitigate the possible load on your system, you may configure the process using several system properties listed below. We strongly recommend reading through the entire process migration process to ensure the optimal configuration for your system.

The migration is configured through a set of properties in Artifactory's `system.properties` file as described below, and essentially, does the following:

- Search for all database records that don't have a SHA-256 value.
- For each such record, find all others in the database with the same SHA1 checksum value
 - If any of them have the SHA-256 calculated already, use that to update all the others
 - If none of them have the SHA-256 calculated already, calculate it and then use that to update all others
- If there are no other records with the same SHA1 value, calculate the SHA-256

First run garbage collection to optimize the migration process

The migration process is complete once all database entries have been populated with SHA-256 values. Since your database may contain entries for artifacts that have been deleted, but have not yet been physically removed by [garbage collection](#), we strongly recommend manually invoking garbage collection before invoking the database migration. Removing deleted artifacts can greatly improve performance and total run time of the migration by reducing the number of downloads it generates.

Configuring the Migration Process

The migration process may be configured through the following [system properties](#).

By default, the migration will run on the primary node, however, using the `forceRunOnNodeId` property described below, you may configure it to run on a secondary node.

Property Name	Function
<code>artifactory.sha2.migration.job.enabled</code>	<p>[Default: false]</p> <p>When true, the process that migrates the database to include SHA-256 checksum for all artifacts will be invoked when the node is restarted.</p>
<code>artifactory.sha2.migration.job.forceRunOnNodeId</code>	<p>[Default: null] Only valid for an HA installation.</p> <p>By default, the migration process runs on the primary node. To run the process on any other node, set this value to the corresponding node's ID (as specified in the <code>node.id</code> property in the nodes <code>\$ARTIFACTORY_HOME/etc/ha-node.properties</code> file).</p> <div style="border: 1px solid green; padding: 5px; margin: 10px 0;"> <p>Running the migration on a dedicated node This gives you the option of dedicating a specific node to run the migration and allocating extra resources allowing it to finish the process faster.</p> </div> <div style="border: 1px solid yellow; padding: 5px; margin: 10px 0;"> <p>Set the property on both the primary and the corresponding secondary node To run the migration process on a secondary node, you need to set this property on BOTH the master node and the corresponding secondary node. Artifactory will still only run the process on the corresponding secondary node.</p> </div>
<code>artifactory.migration.job.dbQueryLimit</code>	<p>[Default: 100]</p> <p>Specifies the number of rows that should be retrieved each time the migration job queries the database for entries that are missing SHA-256 values.</p>
<code>artifactory.migration.job.batchSize</code>	<p>[Default: 10]</p> <p>Artifacts are updated concurrently in batches with new SHA-256 values and then a sleep cycle is initiated. This property specifies the number of artifacts in each batch.</p>
<code>artifactory.sha2.migration.job.queue.workers</code>	<p>[Default: 2]</p> <p>Specifies the number of concurrent threads that should execute actual artifact updates.</p> <p>Each concurrent artifact update may incur a download in order to calculate its SHA-256 checksum. However, the artifact will only be downloaded once, first time a database entry is found for it with no SHA-256 value. Subsequent database entries for the same artifact (which therefore have the same SHA1 value) will reuse the SHA-256 value that was already calculated.</p>
<code>artifactory.migration.job.sleepIntervalMillis</code>	<p>[Default: 5000 milliseconds]</p> <p>Specifies the duration of the sleep cycle which is initiated after each batch of updates.</p>

A sample snippet you can paste into your `artifactory.system.properties` is below, adjust the number of workers as appropriate based on I/O and CPU utilization:

Example artifactory.system.properties snippet

```
##SHA2 Migration block
artifactory.sha2.migration.job.enabled=true
artifactory.sha2.migration.job.queue.workers=5
```

Restart required

For changes to the migration configuration to take effect, you need to restart the instance (or node in the case of an HA installation) that will run it. The default values specified above are set to keep your system performing optimally during the migration process. To speed up the migration process, you may tweak these values (keeping hardware limits in mind), however that may come at a cost of system performance.

Monitoring the Migration Process

Depending on the size of your storage, and the migration parameters you have configured, the migration process may take a long time. To enable easy monitoring of the process, status and error messages are printed into a dedicated log file, `ARTIFACTORY_HOME/logs/sha256_migration.log`. In addition, some messages (process initiation, startup errors) are also logged in the `ARTIFACTORY_HOME/logs/artifactory.log` file.

Configuring Repositories

Overview

Artifactory hosts three types of repository:

- [Local](#)
- [Remote](#)
- [Virtual](#)

Local and remote repositories are true physical repositories, while a virtual repository is actually an aggregation of them used to create controlled domains for search and resolution of artifacts.

To configure repositories, in the **Admin** module, select **Repositories**.

Repositories can be created, deleted, edited, ordered and aggregated.

Single Package Type

When creating any repository, you must specify its package type; this is a fundamental characteristic of the repository and can not be changed later. Once the repository type is set, Artifactory will index artifacts and calculate the corresponding metadata for every package uploaded which optimizes performance when resolving artifacts. Note that virtual repositories can only include repositories of the same type.

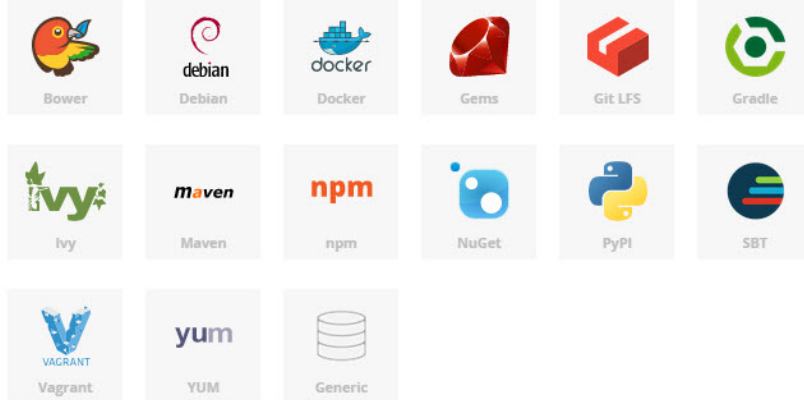
Wrong Package Type

While Artifactory will not prevent you from uploading a package of the wrong type to a repository, we strongly recommend maintaining consistency between the repository type and packages you upload.

if you do upload packages of the wrong type to a repository, Artifactory will not index the package or update the metadata for the repository.

Select Package Type

×



Page Contents

- Overview
- Single Package Type
 - Generic Repositories
- Local Repositories
- Remote Repositories
- Virtual Repositories
 - The Default Virtual Repository (Deprecated)
 - Virtual Resolution Order
- General Resolution Order

Read More

- Common Settings
- Local Repositories
- Remote Repositories
- Smart Remote Repositories
- Virtual Repositories

Generic Repositories

You may define a repository as **Generic** in which case it has no particular type, and you may upload packages of any type. Generic repositories do not maintain separate package indexes. For using a client associated with a specific package type (e.g. yum, gem) you should create a matching repository.

Local Repositories

Local repositories are physical, locally-managed repositories into which you can deploy artifacts.

Artifacts in a local repository can be accessed directly using the following URL:

`http://<host>:<port>/artifactory/<local-repository-name>/<artifact-path>`

Artifactory is deployed with a number of pre-configured local repositories which can be used for internal and external releases, snapshots and plugins.

For full details on configuring local repositories, please refer to [Local Repositories](#).

Remote Repositories

A remote repository serves as a caching proxy for a repository managed at a remote URL (which may itself be another Artifactory remote repository).

Artifacts are stored and updated in remote repositories according to various configuration parameters that control the caching and proxying behavior. You can remove artifacts from a remote repository cache but you cannot actually deploy a new artifact into a remote repository.

Artifacts in a remote repository can be accessed directly using the following URL:

```
http://<host>:<port>/artifactory/<remote-repository-name>/<artifact-path>
```

This URL will fetch a remote artifact to the cache if it has not yet been stored.

In some cases it is useful to directly access artifacts that are already stored in the cache (for example to avoid remote update checks).

To directly access artifacts that are already stored in the cache you can use the following URL:

```
http://<host>:<port>/artifactory/<remote-repository-name>-cache/<artifact-path>
```

Artifactory is deployed with a number of pre-configured, remote repositories which are in common use. Of course you can change these according to the needs of your organization.

Proxy vs. Mirror

A remote repository acts as a **proxy** not as a mirror. Artifacts are not pre-fetched to a remote repository cache. They are only fetched and stored *on demand* when requested by a client.

Therefore, a remote repository should not contain any artifacts in its cache immediately after creation. Artifacts will only be fetched to the cache once clients start working with the remote repository and issuing requests.

For full details on configuring remote repositories please refer to [Remote Repositories](#).

Virtual Repositories

A virtual repository (or "repository group") aggregates several repositories with the same package type under a common URL. The repository is virtual in that you can resolve and retrieve artifacts from it but you cannot deploy artifacts to it.

Generic Virtual Repositories

By their nature, Virtual Repositories whose package type has been specified as **Generic** can aggregate repositories of any type, however generic virtual repositories do not maintain any metadata

The Default Virtual Repository (Deprecated)

Artifactory offers an option to use a global virtual, which contains all local and remote repositories.

By default this option is disabled, to enable the Default Virtual Repository edit the 'artifactory.system.properties' located at \$ARTIFACTORY_HOME/etc and set the following flag to *false*:

```
## Disable the download access to the global 'repo'  
artifactory.repo.global.disabled=false
```

This change requires you restart your Artifactory service.

Once enabled the repository is available at:

```
http://<hostname>:<port>/artifactory/repo
```

Virtual Resolution Order

When an artifact is requested from a virtual repository, the order in which repositories are searched or resolved is local repositories first, then remote repository caches, and finally remote repositories themselves.

Within each of these, the order by which repositories are queried is determined by the order in which they are listed in the configuration as described in [General Resolution Order](#) below.

For a virtual repository, you can see the effective search and resolution order in the **Included Repositories** list view in the **Basic** settings tab. This is particularly helpful when nesting virtual repositories. For more details on configuring a virtual repository please refer to [Virtual Repositories](#).

General Resolution Order

You can set the order in which repositories of each type (local, remote and virtual) are searched and resolved by simply ordering them accordingly within the corresponding section of the **Configure Repositories** page. To set the order you need to add the repositories to the list of selected repositories in the order in which they should be searched to resolve artifacts.

The order in which repositories are searched is also affected by additional factors such as security privileges, include/exclude patterns and policies for handling snapshots and releases.

Common Settings

Overview

Several of the settings are common for local, remote and virtual repositories. These are found in the **Basic Settings** tab of the corresponding **New/Edit** screen under the **General** section. Additional settings may be found in the type-specific section according to the package types specified for the repository.

Common Basic Settings

The screenshot shows the 'New Local Repository' configuration interface. It has three tabs: 'Basic', 'Advanced', and 'Replications'. The 'Basic' tab is selected. Under 'Package Type', 'Maven' is chosen. The 'Repository Key' field is empty. In the 'General' section, 'Repository Layout' is set to 'maven-2-default'. There are fields for 'Public Description' and 'Internal Description'. Below these are 'Include Patterns' and 'Exclude Patterns' sections, each with a 'New Pattern' button and a list of patterns: 'org/apache/**' and 'com/acme/**'. The 'Maven Settings' section includes 'Checksum Policy' (set to 'Verify against client checksums'), 'Maven-Snapshot Version Behavior' (set to 'Unique'), and 'Max Unique Snapshots' (set to '0'). There are also checkboxes for 'Handle Releases', 'Handle Snapshots', and 'Suppress POM Consistency Checks'.

Package Type	The Package Type must be specified when the repository is created, and once set, cannot be changed.
Repository Key	The Repository Key is a mandatory identifier for the repository and must be unique within an Artifactory instance. It cannot begin with a number or contain spaces or special characters. For local repositories we recommend using a "-local" suffix (e.g. "libs-release-local").
Repository Layout	Sets the layout that the repository should use for storing and identifying modules. Artifactory will suggest a layout that corresponds to the package type defined, and index packages uploaded and calculate metadata accordingly.
Public Description	A free text field that describes the content and purpose of the repository.
Internal Description	A free text field to add additional notes about the repository. These are only visible to the Artifactory administrator.

<p>Include and Exclude Patterns</p>	<p>The Include Patterns and the Exclude Patterns fields provide a way to filter out specific repositories when trying to resolve the location of different artifacts.</p> <p>In each field you can specify a list of Ant-like patterns to filter in and filter out artifact queries. Filtering works by subtracting the excluded patterns (default is none) from the included patterns (default is all).</p> <p>Example:</p> <p>Consider that the Include Patterns and Exclude Patterns for a repository are as follows:</p> <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <pre>Include Patterns: org/apache/**,com/acme/** Exclude Patterns: com/acme/exp-project/**</pre> </div> <p>In this case, Artifactory will search the repository for <i>org/apache/maven/parent/1/1.pom</i> and <i>com/acme/project-x/core/1.0/nit-1.0.jar</i> but not for <i>com/acme/exp-project/core/1.1/san-1.1.jar</i> because <i>com/acme/exp-project/**</i> is specified as an Exclude pattern.</p>
<p>JFrog Xray Integration</p>	<p>If Artifactory is connected to an instance of Xray, indicates if the repository is indexed for analysis.</p>

Page Contents

- [Overview](#)
- [Common Basic Settings](#)
 - [Avoiding Security Risks with an Exclude Pattern](#)
 - [Avoiding Performance Issues with an Include Pattern](#)
- [Local and Remote Repositories](#)

Avoiding Security Risks with an Exclude Pattern

Any proprietary artifacts you deploy to Artifactory are stored within local repositories so that they are available for secured and authorized internal use.

Anyone searching for one of your internal artifacts by name will extract it through Artifactory from the local repository.

However, consider what happens if a request for an internal artifact is inadvertently directed *outside* of the organization.

Two examples of how this could happen are:

- there is a simple typo in the requested artifact name
- the developer has requested a snapshot with a version number that does not exist.

In this case, since Artifactory does not find the requested artifact in a local repository, it continues to search through the remote repositories defined in the system. Artifactory will, in fact, search through *all* the remote repositories defined in your system before returning "Not found".

This presents a security risk since any request made on a remote repository may be logged **exposing all details of the query including the full artifact name which may include sensitive business information.**

Best practices using an excludes pattern for remote repositories to avoid security risks

To avoid exposing sensitive business information as described above, we strongly recommend the following best practices:

- The list of remote repositories used in an organization should be managed under a single virtual repository to which all requests are directed
- All internal artifacts should be specified in the **Exclude Pattern** field of the virtual repository (or alternatively, of *each* remote repository) using wildcard characters to encapsulate the widest possible specification of internal artifacts.

Avoiding Performance Issues with an Include Pattern

In a typical scenario, Artifactory will reference large all-purpose repositories such as [JCenter](#) or Maven Central for resolving artifacts.

In addition, Artifactory may reference any number of additional repositories which may host a more specialized and specific set of artifacts.

If Artifactory receives a request for a deterministic set of artifacts (e.g. a specific version of an artifact), then it searches through the different repositories according to its resolution order until the artifact is found.

However, if Artifactory receives a request for a non-deterministic set of artifacts (e.g. all versions of `maven-metadata.xml`) then it must search through *all* of the repositories it references until it can provide a complete response.

In most cases, the majority of artifacts downloaded by an organization will come from one of the large all-purpose repositories, but in non-deterministic requests **performance is downgraded because Artifactory continues to search through all the specialized repositories** before it can return a response.

Best practices using an includes pattern for remote repositories to avoid needless and wasteful search

To avoid performing needless and wasteful search when responding to non-deterministic requests we strongly recommend that all specialized repositories be configured with an appropriate **Include Pattern** specifying only the set of artifacts that the organization might need.

In this case, non-deterministic requests for artifacts that are typically found in general purpose repositories will skip over the specialized repositories thereby improving performance.

Local and Remote Repositories

In addition to the settings above, Local and Remote repositories share the following settings in the type-specific section for relevant package types.

The screenshot shows two settings sections. The first section, 'Maven Snapshot Version Behavior', has a dropdown menu set to 'Unique'. The second section, 'Max Unique Snapshots', has a text input field containing the number '0'. Below these are three checkboxes: 'Handle Releases' (checked), 'Handle Snapshots' (checked), and 'Suppress POM Consistency' (unchecked). Red boxes highlight the 'Unique' dropdown, the 'Max Unique Snapshots' input, and the 'Handle Releases' and 'Handle Snapshots' checkboxes.

Max Unique Snapshots	Specifies the maximum number of unique snapshots of the same artifact that should be stored. Once this number is reached and a new snapshot is uploaded, the oldest stored snapshot is removed automatically. Blank (default) indicates that there is no limit on the number of unique snapshots.
Handle Releases	If set, Artifactory allows you to deploy release artifacts into this repository.
Handle Snapshots	If set, Artifactory allows you to deploy snapshot artifacts into this repository.

Local Repositories

Overview

To configure a local repository, in the **Admin** module, go to **Repositories | Local** and click it to display the **Edit Repository** screen.

Common Basic Settings

The following are fully described in the [Common Settings](#) page.

- Package Type
- Repository Key
- Repository Layout
- Public Description
- Internal Description
- Includes and Excludes Pattern

Page Contents

- Overview
- Common Basic Settings
- Additional Basic Settings
 - Maven, Gradle, Ivy and SBT Repositories
 - Other Repository Types
- Advanced Settings
- Replications
- Pre-defined Local Repositories

Additional Basic Settings

Repositories may have additional **Basic** settings depending on the **Package Type**.

Maven, Gradle, Ivy and SBT Repositories

Maven, Gradle, Ivy and SBT repositories share the same additional **Basic** settings.

Maven

Checksum Policy

Select Checksum Policy...

Maven Snapshot Version Behavior

Select Maven Snapshot Version Behavior...

Max Unique Snapshots

- Handle Releases
- Handle Snapshots
- Suppress POM Consistency

Checksum Policy

Checking the Checksum effectively verifies the integrity of a deployed resource. The **Checksum Policy** determines how Artifactory behaves when a client checksum for a deployed resource is missing or conflicts with the locally calculated checksum.

There are two options:

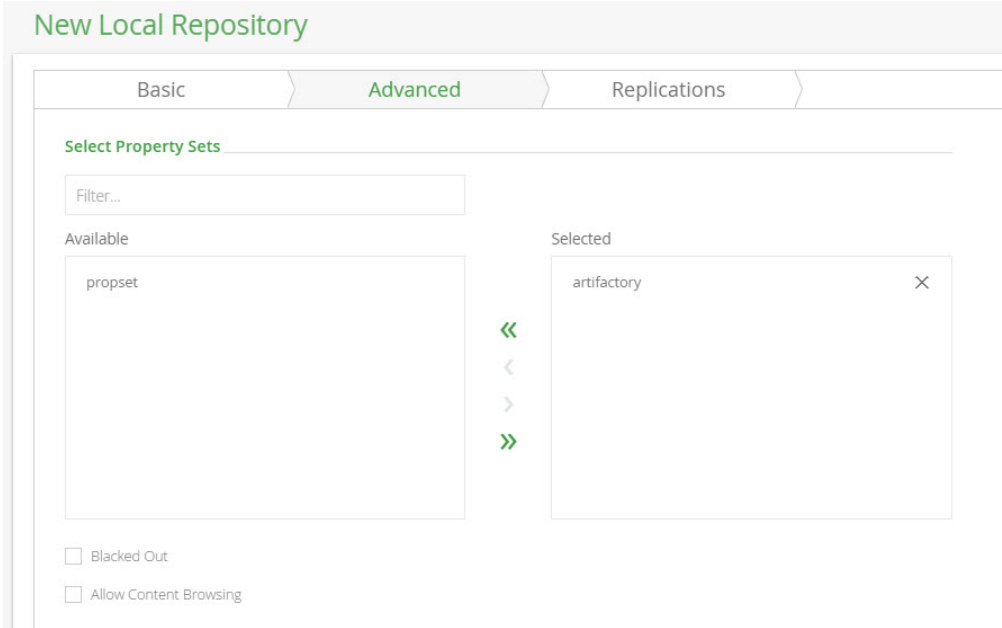
1. **Verify against client checksums** (default) - If a client has not sent a valid checksum for a deployed artifact then Artifactory will return a 404 (not found) error to a client trying to access that checksum. If the client has sent a checksum, but it conflicts with the one calculated on the server then Artifactory will return a 409 (conflict) error until a valid checksum is deployed.
2. **Trust server generated checksums** - Artifactory will not verify checksums sent by clients and will trust the server's locally calculated checksums. An uploaded artifact is immediately available for use, but integrity might be compromised.

<p>Maven Snapshot Version Behavior</p>	<p>Artifactory supports centralized control of how snapshots are deployed into a repository, regardless of end user-specific settings. This can be used to guarantee a standardized format for deployed snapshots within your organization. There are three options:</p> <ol style="list-style-type: none"> 1. Unique: Uses a unique, time-based version number 2. Nonunique: Uses the default self-overriding naming pattern: <code>artifactID-version-SNAPSHOT.type</code> 3. Deployer: Uses the format sent by the deployer as is. <div style="border: 1px solid green; padding: 10px; margin-top: 10px;"> <p>Maven 3 Only Supports Unique Snapshots</p> <p>Maven 3 has dropped support for resolving and deploying non-unique snapshots. Therefore, if you have a snapshot repository using non-unique snapshots, we recommend that you change your Maven snapshot policy to 'Unique' and remove any previously deployed snapshots from this repository.</p> <p>The unique snapshot name generated by the Maven client on deployment cannot help in identifying the source control changes from which the snapshot was built and has no relation to the time sources were checked out. Therefore, we recommend that the artifact itself should embed the revision/tag (as part of its name or internally) for clear and visible revision tracking. Artifactory allows you to tag artifacts with the revision number as part of its Build Integration support.</p> </div>
<p>Max Unique Snapshots</p>	<p>Specifies the maximum number of unique snapshots of the same artifact that should be stored. Once this number is reached and a new snapshot is uploaded, the oldest stored snapshot is removed automatically.</p> <p>A value of 0 (default) indicates that there is no limit on the number of unique snapshots.</p>
<p>Handle Releases</p>	<p>If set, Artifactory allows you to deploy release artifacts into this repository.</p>
<p>Handle Snapshots</p>	<p>If set, Artifactory allows you to deploy snapshot artifacts into this repository.</p>
<p>Suppress POM Consistency</p>	<p>When deploying an artifact to a repository, Artifactory verifies that the value set for <code>groupId:artifactId:version</code> in the POM is consistent with the deployed path.</p> <p>If there is a conflict between these then Artifactory will reject the deployment. You can disable this behavior by setting this checkbox.</p>

Other Repository Types

For other type-specific repository configuration, please refer to the corresponding repository page under [Artifactory Pro](#).

Advanced Settings



Select Property Sets	Defines the property sets that will be available for artifacts stored in this repository.
Blacked Out	If set, Artifactory ignores this repository when trying to resolve artifacts. The repository is also not available for download or deployment of artifacts.
Allow Content Browsing	<p>If set, allows you to view file contents (e.g., Javadoc browsing, HTML files) directly from Artifactory.</p> <div style="border: 1px solid red; padding: 5px;"> <p>Security When content browsing is allowed we recommend strict content moderation to ensure that any uploaded content does not compromise security (for example, cross-site scripting attacks)</p> </div>

Replications

The **Replications** tab lets you define and edit replication settings for the repository. For details, please refer to [Repository Replication](#).

Pre-defined Local Repositories

Artifactory comes with a set of pre-defined local repositories, which reflect best practices in binary repository management as follows:

<i>libs-release-local</i>	Your code releases
<i>libs-snapshot-local</i>	Your code snapshots
<i>ext-release-local</i>	Manually deployed 3rd party libs (releases)
<i>ext-snapshot-local</i>	Manually deployed 3rd party libs (shapshots)
<i>plugins-release-local</i>	Your and 3rd party plugins (releases)

<i>plugins-snapshot-local</i>	Your and 3rd party plugins (snapshots)
-------------------------------	--

Remote Repositories

Overview

To configure a remote repository, in the **Admin** module, go to **Repositories | Remote** and click it to display the **Edit Repository** screen.

Common Basic Settings

The following are fully described in the [Common Settings](#) page.

- [Package Type](#)
- [Repository Key](#)
- [Public Description](#)
- [Internal Notes](#)
- [Includes and Excludes Pattern](#)

Additional Basic Settings

URL	The URL for the remote repository. Currently only HTTP and HTTPS URLs are supported.
Offline	If set, this repository will be considered offline and no attempts will be made to fetch artifacts from it. For more details, please refer to Single Repository Offline below.

Page Contents

- [Overview](#)
- [Common Basic Settings](#)
- [Additional Basic Settings](#)
- [Type-Specific Basic Settings](#)
 - [Maven, Gradle, Ivy and SBT Repositories](#)
 - [Other Repository Types](#)
- [Handling Offline Scenarios](#)
 - [Single Repository Offline](#)
 - [Global Offline Mode](#)

Read More

- [Managing Proxies](#)
- [Advanced Settings](#)

Type-Specific Basic Settings

Repositories may have additional **Basic** settings depending on the **Package Type**.

Maven, Gradle, Ivy and SBT Repositories

Maven Settings

Checksum Policy

Generate if absent

Max Unique Snapshots

0

Eagerly Fetch Jars

Suppress POM Consistency

Eagerly Fetch Sources

List Remote Folder Items

Handle Releases

Handle Snapshots

Checksum Policy	<p>Checking the Checksum effectively verifies the integrity of a deployed resource. The Checksum Policy determines how Artifactory behaves when a client checksum for a remote resource is missing or conflicts with the locally calculated checksum.</p> <p>There are four options:</p> <ol style="list-style-type: none">Generate if absent (default): Artifactory attempts to retrieve the remote checksum, If it is not found, Artifactory will automatically generate one and fetch the artifact. If the remote checksum does not match the locally calculated checksum, the artifact will not be cached and the download will fail.Fail: If the remote checksum does not mach the locally calculated checksum, or is not found, the artifact will not be cached and the download will fail.Ignore and generate: Artifactory ignores the remote checksum and only uses the locally generated one. As a result, remote artifact retrieval never fails, however integrity of the retrieved artifact may be compromised.Ignore and Pass-thru: Artifactory stores and passes through all remote checksums (even if they do not match the locally generated one). If a remote checksum is not found, Artifactory generates one locally. As a result, remote resource retrieval never fails, however integrity of the retrieved artifact may be compromised, and client side checksum validation (as performed by Maven, for example) will fail.
Max Unique Snapshots	Please refer to Max Unique Snapshots under Local Repositories.
Eagerly Fetch Jars	When set, if a POM is requested, Artifactory attempts to fetch the corresponding jar in the background. This will accelerate first access time to the jar when it is subsequently requested.
Suppress POM Consistency	<p>By default, Artifactory keeps your repositories healthy by refusing POMs with incorrect coordinates (path). If the <code>groupId:artifactId:version</code> information inside the POM does not match the deployed path, Artifactory rejects the deployment with a "409 Conflict" error.</p> <p>You can disable this behavior by setting the Suppress POM Consistency checkbox.</p>
Eagerly Fetch Sources	When set, if a binaries jar is requested, Artifactory attempts to fetch the corresponding source jar in the background. This will accelerate first access time to the source jar when it is subsequently requested.

Handle Releases	Please refer to Handle Releases under Local Repositories.
Handle Snapshots	Please refer to Handle Snapshots under Local Repositories.

Other Repository Types

For other type-specific repository configuration, please refer to the specific repository page under [Artifactory Pro](#).

Handling Offline Scenarios

Artifactory supports offline repository management at two levels:

- **Single Repository:** One or more specific remote repositories need to be offline.
- **Global:** The whole organization is disconnected from remote repositories

Single Repository Offline

If a remote repository goes offline for any reason, Artifactory can be configured to ignore it by setting the **Offline** checkbox. In this case, only artifacts from this repository that are already present in the cache are used. No further attempt will be made to fetch remote artifacts.

Global Offline Mode

This is common in organizations that require a separate, secured network and are disconnected from the rest of the world (for example, military or financial institutions) .

In this case, remote repositories serve as caches only and do not proxy remote artifacts.

You can enable Global Offline Mode by setting the corresponding checkbox in the **Admin** tab under **Configuration | General**.

General Settings

Server Name

Arti4-Demo

Custom URL Base

File Upload Max Size *

100

Date Format *

dd-MM-yy HH:mm:ss z

Global Offline Mode

Managing Proxies

Overview

In corporate environments it is often required to go through a proxy server to access remote resources.

Artifactory supports several types of network proxy including NTLMv2 (if running on Linux you may use NTLMv2 only with CNTLM).

Page Contents

- [Overview](#)
- [Defining Proxies](#)

Defining Proxies

To create a new proxy definition, in the **Admin** module go to **Configuration | Proxies** and click the "New" button.

Fields that are not required by the proxy may be left blank (for example, if you are not using authentication credentials or with an NTLM proxy you may leave the **Username** and **Password** fields blank).

New Proxy

Proxy Settings

Proxy Key *

System Default

Host *

Port *

User Name

Password

NT Host

NT Domain

Redirecting Proxy Target Hosts

Proxy Key	The unique ID of the proxy.
System Default	When set, this proxy will be the default proxy for new remote repositories and for internal HTTP requests issued by Artifactory. When you set this checkbox, Artifactory displays a confirmation message and offers to apply the proxy setting also to existing remote repository configurations.
Host	The name of the proxy host.
Port	The proxy port number.
Username	The proxy username when authentication credentials are required.
Password	The proxy password when authentication credentials are required.
NT Host	The computer name of the machine (the machine connecting to the NTLM proxy).
NT Domain	The proxy domain/realm name.
Redirecting Proxy target Hosts	An optional list of newline or comma separated host names to which this proxy may redirect requests. The credentials defined for the proxy are reused by requests redirected to all of these hosts.

Using proxies

Artifactory only accesses a remote repository through a proxy if one is selected in the **Network** section of the **Advanced** settings for a remote repository.

Whether this has been set manually, or by setting a **System Default** proxy as described [above](#), you can override this by removing the **Proxy** setting for any specific repository.

In this case, Artifactory will access the specific repository without going through a proxy.

Advanced Settings

Overview

The advanced settings for a remote repository configure network access behavior, cache management and several other parameters related to remote repository access.

To access the advanced settings, in the **Edit Remote Repository** screen select the **Advanced** tab.

Remote Credentials

Remote Authentication

Username

Password

Test

SSL / TLS Certificate

Username	The username that should be used for HTTP authentication when accessing this remote proxy.
----------	--

Password	The password that should be used for HTTP authentication when accessing this remote proxy.
SSL/TLS Certificate	The SSL/TLS certificate this repository should use for authentication to the remote resource for which it is a proxy.

Page Contents

- Overview
- Remote Credentials
- Network Settings
- Cache Settings
 - Zapping Caches
- Select Property Sets
- Other Settings

Network Settings

Network

Proxy	Local Address
<input type="text" value="Select Proxy..."/>	<input type="text"/>
Socket Timeout	Query Params
<input type="text" value="15000"/>	<input type="text"/>
<input type="checkbox"/> Lenient host authentication	<input type="checkbox"/> Cookie management

Proxy	If your organization requires you to go through a proxy to access a remote repository, this parameter lets you select the corresponding Proxy Key . For more details on setting up proxies in Artifactory please refer to Managing Proxies .
Local Address	When working on multi-homed systems, this parameter lets you specify which specific interface (IP address) should be used to access the remote repository. This can be used to ensure that access to the remote repository is not blocked by firewalls or other organizational security systems.
Socket Timeout	The time that Artifactory waits (for both a socket and a connection) before giving up on an attempt to retrieve an artifact from a remote repository. Upon reaching the specified Socket Timeout Artifactory registers the repository as "assumed offline" for the period of time specified in Assumed Offline Limit .
Query Params	A custom set of parameters that should automatically be included in all HTTP requests to this remote repository. For example, <code>param1=value1&param2=value2&param3=value3</code>
Lenient Host Authentication	When set, allows using the repository credentials on any host to which the original request is redirected.
Cookie Management	When set, the repository will allow cookie management to work with servers that require them.

Using Oracle Maven Repository

To use [Oracle Maven Repository](#):

- Set your Oracle credentials in **Username** and **Password** of the **Remote Credentials**
- Set **Lenient Host Authentication**

- Set **Enable Cookie Management**.

Cache Settings

Artifactory stores artifacts retrieved from a remote repository in a local cache. The **Cache Settings** specify how to manage cached artifacts.

Caching Maven artifacts

Caching for Maven artifacts is only applicable to snapshots since it is assumed that releases never change.

Cache

Unused Artifacts Cleanup Period

Retrieval Cache Period

Assumed Offline Period

Missed Retrieval Cache Period

Unused Artifacts Cleanup Period	<p>Many cached artifacts in Artifactory remote repository storage are actually unused by any current projects in the organization. This parameter specifies how long an unused artifact will be stored before it is removed. Once reaching this period Artifacts will be removed in the next invocation of cleanup. For more details please refer Cleanup Unused Cached Artifacts in Regular Maintenance Operations.</p> <p>Leaving the field empty (default) means that the artifact is stored indefinitely.</p>
Metadata Retrieval Cache Period	<p>Defines how long before Artifactory checks for a newer version of a requested artifact in a remote repository. A value of 0 means that Artifactory will always check for a newer version.</p> <div data-bbox="261 1161 1481 1350" style="border: 1px solid black; padding: 10px;"> <p>On which file types does this parameter work? This setting refers to artifacts that expire after a period of time (e.g. metadata files such as <i>maven-metadata.xml</i>, <i>npm package.json</i> or Docker <i>manifest.json</i> etc.).</p> <p>Note that most artifacts that are downloaded do not change (e.g. release versions), therefore this setting has no effect on them.</p> </div>
Assumed Offline Period	<p>In case of a connection error, this parameter specifies how long Artifactory should wait before attempting an online check in order to reset the offline status. A value of 0 means that the repository is never assumed offline and Artifactory will always attempt to make the connection when demanded.</p>
Missed Retrieval Cache Period	<p>If a remote repository is missing a requested artifact, Artifactory will return a "404 Not found" error. This response is cached for the period of time specified by this parameter. During that time, Artifactory will not issue new requests for the same artifact. A value of 0 means that the response is not cached and Artifactory will always issue a new request when demanded.</p>

Zapping Caches

"Zapping" a cache means forcing the Retrieval Cache Period and Missed Retrieval Cache Period to time out. To "zap" a cache, in the **Artifacts** module **Tree** browser,

Select the repository cache you wish to "zap" and click **Zap caches** in the right-click menu or **Actions** drop-down menu.

Actions

- Refresh
- Copy Content
- Move Content
- Watch
- Zap Cache
- Delete Versions
- Delete Content

Select Property Sets

Defines the property sets that will be available for artifacts stored in this repository.

Other Settings

Others

- Blacked Out ?
- Allow Content Browsing ?
- Store Artifacts Locally ?
- Synchronize Properties ?
- Bypass HEAD Requests ?
- Block Mismatching Mime Types ?

Override Default Blocked Mime Types

Blacked out	If set, Artifactory ignores this repository when trying to resolve artifacts. The repository is also not available for download or deployment of artifacts.
-------------	---

Allow content browsing	<p>When set, allows Artifactory users to browse the internal contents of archives (for example, browsing specific Javadoc files from within a Javadoc archive).</p> <div style="border: 1px solid red; padding: 5px; margin-top: 10px;"> <p>Care When archive browsing is allowed, strict content moderation should be employed to ensure malicious users do not upload content that may compromise security (e.g. cross-site scripting attacks)</p> </div>
Store artifacts locally	<p>When set, Artifactory artifacts from this repository will be cached locally. If not set, direct repository-to-client streaming is used.</p> <div style="border: 1px solid green; padding: 5px; margin-top: 10px;"> <p>When might you use direct repository-to-client streaming? If your organization has multiple servers connected over a high speed LAN, you may have one instance of Artifactory caching data on a central storage facility with additional instances of Artifactory running on other servers. In this case, it makes sense for the additional instances of Artifactory to act as satellite pass-through servers rather than have them duplicate the cached data within their own environments.</p> </div>
Synchronize properties	<p>When set, synchronizes properties of artifacts retrieved from a remote instance of Artifactory.</p>
Bypass HEAD Requests	<p>When set, Artifactory will not send a HEAD request to the remote resource before downloading an artifact for caching</p>
Block Mismatching Mime Types	<p>When set, artifacts will fail to download if a mismatch is detected between the requested and received mime type, according to a list specified in the <i>system.properties</i> file under <code>blockedMismatchingMimeTypes</code>. You can override this setting by adding mime types to the override list below.</p>
Override Default Blocked Mime Types	<p>The set of mime types that should override the Block Mismatching Mime Types setting.</p>
Propagate Query Params	<p>When set, if query params are included in the request to Artifactory, they will be passed on to the remote repository.</p> <div style="border: 1px solid blue; padding: 5px; margin-top: 10px;"> <p>Generic Repositories Only This setting is only available for Generic type repositories.</p> </div>

Smart Remote Repositories

Overview

A smart remote repository is a remote repository that proxies a repository from another instance of Artifactory. In addition to the usual benefits of [remote repositories](#), smart remote repositories offer several additional benefits:

Reported download statistics

Artifactory maintains download statistics for repositories so you are able to evaluate if artifacts are still being used and manage your cleanup policies. When you proxy a repository in another instance of Artifactory, and cache an artifact downloaded from the other instance, the distant Artifactory is not aware if users on your end continue to use the artifact (downloading it from your local cache), and may end up cleaning up the original artifact. An Artifactory Smart Remote Repository lets you notify the distant instance whenever a cached artifact is downloaded, so it can update an internal counter for remote downloads.

Page Contents

- [Overview](#)
- [Configuration](#)
- [Remote List Browsing](#)

Download statistics may vary between Artifactory instances

Downloads are only reported through the proxy chain from the time this option is set, so the actual download statistics reported for an artifact may be different in the local Artifactory instance compared the numbers reported in the remote Artifactory instance.

Synchronized properties

When you proxy a repository in another instance of Artifactory and cache an artifact downloaded from it, you may not be aware of changes that may have been made to the original artifact's properties if they are done after you cache it. By synchronizing properties, any changes to artifact properties in the remote instance are propagated to your cached instance of the artifact.

Remote repository browsing

You can browse the contents of the repository in the remote Artifactory instance for all package types, even if none have been cached in your instance of Artifactory.

Source absence detection

When viewing a cached artifact, Artifactory will indicate if the original artifact in the remote instance has been deleted. This gives you an opportunity to copy the artifact over from your remote repository cache to a local repository in case you need to maintain access to it.

update an internal counter for remote downloads.

Configuration

To create a Smart Remote Repository, set the repository **URL** to point to a repository in another instance of Artifactory.

Repository URL must be prefixed with `api/<type>`

To accommodate different packaging format clients, for several repository types, when accessing the repository through Artifactory, the repository URL must be prefixed with `api/<type>` in the path.

For example,

```
http://ARTIFACTORY_URL/api/<package type>/<repository key>
```

Or, if you are using Artifactory SaaS the URL would be:

```
https://<server name>.jfrog.io/<server name>/api/<package type>/<repository key>
```

The prefix is required for the following repository types:

Type	Prefix
Bower	api/bower
CocoaPods	api/pods


Docker	api/docker
NuGet	api/nuget
Npm	api/npm
PyPI	api/pypi
RubyGems	api/gems
Vagrant	api/vagrant
PHP Composer	api/composer
Chef	api/chef
Puppet	api/puppet

New Remote Repository

Basic > Advanced > Replications

Package Type *

maven

Maven 

Repository Key *

smart-remote

URL *

http://[redacted]/artifactory

Test

Once you have finished entering the URL and move to another field, Artifactory automatically detects that the remote URL is on another instance of Artifactory and displays a dialog where you can configure the behavior of your smart remote repository.

Note also that the package type icon is overlaid with an Artifactory logo to indicate a smart remote repository.

Artifactory Remote Repository Detected

Since the remote repository at the URL you specified is hosted by another instance of Artifactory, you may configure some additional capabilities for your repository:

Report Statistics

If set, download statistics for the artifact at the remote Artifactory instance will be updated each time a cached item is downloaded from your repository.

Sync Properties

If set, properties for artifacts that have been cached in this repository will be updated if they are modified in the artifact hosted at the remote Artifactory instance.

List Remote Folder Items

If set, Artifactory lets you navigate the contents of the repository at the remote Artifactory instance, for all package types, even if the artifacts have not been cached in this repository.

Source Absence Detection

When viewing a cached item in this repository, you will see an indication if it has been deleted from the repository at the remote Artifactory instance.

These settings are also available in the Edit Repository screen.

Do not show this message again

OK

Report Statistics	<p>If set, Artifactory will notify the remote instance whenever an artifact in the Smart Remote Repository is downloaded locally so the it can update its download counter.</p> <p>Note that if this option is not set, there may be a discrepancy between the number of artifacts reported to have been downloaded in the different Artifactory instances of the proxy chain.</p>
Sync Properties	<p>If set, properties for artifacts that have been cached in this repository will be updated if they are modified in the artifact hosted at the remote Artifactory instance.</p> <p>The trigger to synchronize the properties is download of the artifact from the remote repository cache of the local Artifactory instance.</p>
List Remote Folder Items	<p>If set, enables Remote List Browsing.</p>
Source Absence Detection	<p>If set, Artifactory displays an indication on cached items if they have been deleted from the corresponding repository in the remote Artifactory instance.</p>

You can modify these settings at any time from the **Edit Repository** screen.

Edit smart-remote Repository

Basic

Advanced

Replications

Package Type *



Repository Key *

smart-remote

URL *

http://[redacted]/artifactory/libs-release-local

Test

General

Repository Layout

maven-2-default

Remote Layout Mapping

Select Remote Layout

Smart Remote Repository

- Report Statistics ?
- Sync Properties ?
- List Remote Folder Items ?
- Source Absence Detection ?

Remote List Browsing

When **List Remote Folder Items** is checked for a repository, Artifactory lets you navigate the contents of the repository at the remote Artifactory instance, for all package types, even if the artifacts have not been cached in the repository defined in your instance of Artifactory.

Virtual Repositories

Overview

To simplify access to different repositories, Artifactory allows you to define a virtual repository which is a collection of local, remote and other virtual repositories accessed through a single logical URL.

A virtual repository hides the access details of the underlying repositories letting users work with a single, well-known URL. The underlying participating repositories and their access rules may be changed without requiring any client-side changes.

Page Contents

- Overview
- Basic Settings
 - Nesting
 - Using Includes and Excludes Patterns
- Deploying to a Virtual Repository
- Advanced Settings
 - Maven, Gradle, Ivy and SBT Repositories
- Pre-defined Repositories

Basic Settings


The following are fully described in the [Common Settings](#) page.

- Package Type
- Repository Key
- Repository Layout
- Public Description
- Internal Description
- Includes and Excludes Pattern

New Virtual Repository

Basic Advanced

Package Type *


Maven

Repository Key *

General

Repository Layout

Public Description

Internal Description

Include Pattern

Exclude Pattern

In addition, in the **Repositories** section of the **Basic** settings screen you select the **Available Repositories** you want to include in the new virtual repository and move them to the **Selected Repositories** list.

This list can be re-ordered by dragging and dropping within the **Selected Repositories** list.

Repositories

Filter...

Available Repositories

- libs-release
- libs-snapshot
- p2
- p2-virtual
- plugins-release
- plugins-snapshot
- test-p2
- vri-terra-cotta

Selected Repositories

- libs-release-local
- ext-snapshot-local
- remote-repos

Included Repositories

libs-release-local
ext-snapshot-local
java.net.m1
repo1
gradle-libs

The **Included Repositories** section displays the effective list of actual repositories included in this virtual repository. If any of the available repositories you have selected are themselves virtual repositories, then the **Included Repositories** section will display the local and remote repositories included within them. The **Included Repository** list is automatically updated in case any of the nested virtual repositories change.

The search/resolution order when requesting artifacts from a virtual repository is always:

1. Local repositories
2. Remote repository caches
3. Remote repositories themselves.

The order within these categories is controlled by the order they are presented in the **Selected Repositories** list.

Nesting

Nesting is a unique feature in Artifactory and facilitates more flexibility in using virtual repositories.

You should take care not to create an "infinite loop" of nested repositories. Artifactory analyzes the internal composition of virtual repositories and will issue a warning if the virtual repository can not be resolved due to invalid nesting.

Using Includes and Excludes Patterns

The ability to define and **Includes Pattern** and an **Excludes Pattern** for virtual repositories (especially when nesting is used) provides a powerful tool you can use to manage artifact requests in your organization.

For example, your organization may have its own artifacts which are hosted both internally in a local repository, but also in a remote repository. For optimal performance, you would want these artifacts to be accessed from the local repository rather than from the remote one. To enforce this policy, you can define a virtual repository called "remote-repos" which includes the full set of remote repositories accessed by your organization, and then specify an Excludes Pattern with your organization's groupId. In this way, any attempt to access your internal artifact from a remote repository would be rejected.

Consider another example in which you wish to define a virtual repository for your developers, however you wish to keep certain artifacts hidden from them. This could be achieved by defining an **Excludes Pattern** based on groupId, source or version.

Deploying to a Virtual Repository

From version 4.2, Artifactory supports deploying artifacts to a virtual repository. For example you can now use `docker push`, `npm publish`,

NuGet push, gem push Artifactory's REST API and more to deploy packages to a virtual repository.

For more details, please refer to [Deploying Artifacts](#).

Advanced Settings

New Virtual Repository

Basic Advanced

Artifactory Requests Can Retrieve Remote Artifacts

Cleanup Repository References in POMs

Discard active references

Key-Pair

No key-pairs are currently configured. You can add new key-pairs [here](#).

Artifactory Requests Can Retrieve Remote Artifacts

An Artifactory instance may request artifacts from a virtual repository in another Artifactory instance. This checkbox specifies whether the virtual repository should search through remote repositories when trying to resolve an artifact requested by another Artifactory instance. For example, you can use this feature when Artifactory is deployed in a mesh (grid) architecture, and you do not want all remote instances of Artifactory to act as proxies for other Artifactory instances.

Maven, Gradle, Ivy and SBT Repositories

In addition to the above checkbox, these repository types offer the following **Advanced** settings:

<p>Cleanup Repository References in POMs</p>	<p>Public POMs may include direct references to external repositories. If either of the below code samples are present in the POM, Maven dynamically adds an external repository URL to the build which circumvents Artifactory.</p> <div style="border: 1px dashed blue; padding: 10px; margin: 10px 0;"> <pre><project><repositories><repository> or <project><pluginRepositories><pluginRepository></pre> </div> <p>A client side solution for this is to use mirrorOf. For details please refer to Additional "Mirror-any" Setup.</p> <p>This setting gives you the ability to ensure Artifactory is the sole provider of Artifacts in your system by automatically cleaning up the POM file. The three values available for this setting are:</p> <table border="1" data-bbox="365 520 1485 840"> <tr> <td data-bbox="365 520 568 636">Discard Active References</td> <td data-bbox="574 520 1485 636">Removes repository elements that are declared directly under project or under a profile in the same POM that is <code>activeByDefault</code></td> </tr> <tr> <td data-bbox="365 644 568 747">Discard Any References</td> <td data-bbox="574 644 1485 747">Removes all repository elements regardless of whether they are included in an active profile or not</td> </tr> <tr> <td data-bbox="365 756 568 840">Nothing</td> <td data-bbox="574 756 1485 840">Does not remove any repository elements declared in the POM</td> </tr> </table>	Discard Active References	Removes repository elements that are declared directly under project or under a profile in the same POM that is <code>activeByDefault</code>	Discard Any References	Removes all repository elements regardless of whether they are included in an active profile or not	Nothing	Does not remove any repository elements declared in the POM
Discard Active References	Removes repository elements that are declared directly under project or under a profile in the same POM that is <code>activeByDefault</code>						
Discard Any References	Removes all repository elements regardless of whether they are included in an active profile or not						
Nothing	Does not remove any repository elements declared in the POM						
<p>Key Pair</p>	<p>A named key-pair to use for automatically signing artifacts.</p> <p>Please refer to WebStart and Jar Signing.</p>						

Pre-defined Repositories

Artifactory comes with a set of pre-defined virtual repositories, which reflect binary management best practices as follows.

remote-repos	Aggregation of all remote repositories
lib-releases	libs-releases-local, ext-releases and remote-repos
plugins-releases	plugins-releases-local, ext-releases and remote-repos
libs-snapshots	libs-snapshots-local, ext-snapshots-local, remote-repos
plugins-snapshots	plugins-snapshots-local, ext-snapshots-local, remote-repos

Configuring Security

Overview

Artifactory's security model offers protection at several levels. It allows you to do the following:

- Assign role-based or user-based permissions to areas in your repositories (called Permission Targets)
- Allow sub-administrators for Permission Targets
- Configure LDAP out-of-the-box
- Prevent clear text in Maven's `settings.xml` file
- Inspect security definitions for a single artifact or folder and more.

Artifactory's security is based on Spring Security and can be extended and customized.

This section explains the strong security aspects and controls offered by Artifactory.

General Configuration

Artifactory provides several system-wide settings to control access to different resources. These are found under **Security | General** in the **Administration** tab.

General Security Settings

- Allow Anonymous Access
- Prevent Anonymous Access to Build Related Info
- Hide Existence of Unauthorized Resources [?](#)

Password Encryption Policy [?](#)

SUPPORTED

Page Contents

- Overview
- General Configuration
 - Allow Anonymous Access
 - Prevent Anonymous Access to Build Related Info
 - Hide Existence of Unauthorized Resources
 - Password Encryption Policy
 - User Lock and Login Suspension
 - Temporary Login Suspension
 - User Account Locking
 - Unlocking User Accounts
 - Password Expiration Policy
 - Managing API Keys

- Passwords Encryption

Read More

- Managing Users
- Managing Permissions
- Centrally Secure Passwords
- Master Key Encryption
- Managing Security with LDAP
- Managing Security with Active Directory
- Managing Certificates
- Using a Self-Signed Certificate
- Access Tokens
- Access Log

Allow Anonymous Access

Artifactory provides a detailed and flexible permission-based system to control users' access to different features and artifacts.

However, Artifactory also supports the concept of "Anonymous Access" which controls the features and artifacts available to a user who has not logged in.

This is done through an "Anonymous User" which comes built-in to Artifactory with a default set of permissions.

Anonymous access may be switched on (default) or off using the **Allow Anonymous Access** setting under **Security General Settings** in the **Administration** module.

You can modify the set of permissions assigned to the "Anonymous User" just like you would for any other user, and this requires that **Allow Anonymous Access** is enabled.

Prevent Anonymous Access to Build Related Info

This setting gives you more control over anonymous access, and allows you to prevent anonymous users from accessing the **Build** module where all information related to builds is found, even when anonymous access is enabled.

Hide Existence of Unauthorized Resources

When a user tries to access a resource for which he is not authorized, Artifactory's default behavior is to indicate that the resource exists but is protected.

For example, an anonymous request will result in a request for authentication (401), and a request by an unauthorized authenticated user will simply be denied (403).

You can configure Artifactory to return a 404 (instead of 403) - Not Found response in these cases by setting **Hide Existence of Unauthorized Resources** under **Security | General** in the **Administration** module.

Password Encryption Policy

Artifactory provides a unique solution to support encrypted passwords through the **Password Encryption Policy** setting as follows:

Supported	Artifactory can receive requests with an encrypted password but will also accept requests with a non-encrypted password (default)
Required	Artifactory requires an encrypted password for every authenticated request
Unsupported	Artifactory will reject requests with encrypted password


For more details on why Artifactory allows you to enforce password encryption please refer to [Centrally Secure Passwords](#).

User Lock and Login Suspension

User Lock

Lock User After Exceeding Max Failed Login Attempts

Max Failed Login Attempts

 **Unlock All Users**

User account locking and temporary login suspension are two mechanisms employed by Artifactory to prevent identity theft via brute force attack.

Temporary Login Suspension

Temporary login suspension means that when a login attempt fails due to incorrect authentication credentials being used, Artifactory will temporarily suspend that user's account for a brief period of time during which Artifactory ignores additional login attempts. If login attempts fail repeatedly, Artifactory will increase the suspension period each time until it reaches a maximum of 5 seconds.

User Account Locking

In addition to temporary login suspension, you can configure Artifactory to lock a user's account after a specified number of failed login attempts. This is enabled by checking "Lock User After Exceeding Max Failed Login Attempts", and specifying the **Max Failed Login Attempts** field. Users who get locked out of their account because they have exceeded the maximum number of failed login attempts allowed (as specified in **Max Failed Login Attempts**) must have an administrator access to unlock their account.

Unlocking User Accounts


An Artifactory administrator can unlock all locked-out users using the "Unlock All Users" button under **Security General Configuration** screen where user locking is configured. An administrator can also unlock a specific user or a group of users in the **Security Module** under **User Management**.

Users Management

28 Users (1 Selected) New

Filter by Name or Email

Delete Unlock Page 1 of 2

Name	Email	Realm	Related Groups	Related Permissions	Admin	Locked	Last Login
aaa	rhojug@gmail.com	internal	1 readers	1 Anything			11-01-16 10:30:28 UTC

Through the REST API, an administrator can unlock a [single user](#), a [group of users](#) or [all locked-out users at once](#).

Password Expiration Policy

Artifactory lets an admin user enforce a password expiration policy that forces all users to change their passwords at regular intervals. When the password expiration policy is enforced, users who do not within the specified time interval will be locked out of their accounts until they change their password.

Password Expiration Policy

Enable Password Expiration Policy

Password Expires Every (Days)

60

Send Mail Notification Before Password Expiration

Force Password Expiration For All Users

Enable Password Expiration Policy	When checked, password expiration policy is enabled.
Password Expires Every (Days)	Specifies how frequently all users must change their password.
Send Mail Notification Before Password Expiration	When checked, users receive an email notification a few days before their password expires.
Force Password Expiration For All Users	Forces all passwords to expire. All users will have to change their password at next login.

Managing API Keys

As an admin user, you can revoke all the API keys currently defined in the system under **Security | General** in the **Administration** module.

API Keys Management

Revoke API Keys For All Users

To revoke all API keys in the system, click "Remove API Keys for All Users".

To revoke a specific user's API key, navigate to **Administration** module >> **Security | Users** and select the relevant user to edit . Once in the edit screen one of the available actions is "Revoke API key"

Once you revoke an API key, any REST API calls using that API key will no longer work. The user will have to create new API key and update any scripts that use it.

Passwords Encryption

Different configuration files in Artifactory may include password information stored in plain text.

If you click "Encrypt", Artifactory will generate a master encryption key and encrypt all passwords.

Managing Users

Overview

You can manage access to repositories by defining users, assigning them to groups and setting up roles and permissions which can be applied to both users and groups.

Creating and Editing Users

To manage users who can access repositories in your system, in the **Admin** module, select **Security | Users**

Name	Email	Realm	Related Groups	Admin	Locked	Last Login
admin		internal	2 other-group, yron			20-05-15 14:53:27 UTC
admin		internal	2 other-group, yron			01-07-16 10:56:52 UTC
anonymous		internal	2 other-group, readers			
elery		internal				16-10-15 16:16:25 UTC
elma		internal	2 other-group, yron			21-07-15 17:28:12 UTC
elma-deployer		internal	3 all, other-group, deployers, readers, yron			21-07-15 08:34:08 UTC
elma-master		internal	3 other-group, readers, yron			21-07-15 08:34:08 UTC
elma@1@big.com		saml Check external status	4 other-group, readers, yron, managers 2			
elvat		internal	1 managers 1			
elvg		saml Check external status				12-03-16 22:11:12 UTC
elvst		internal	2 other-group, readers			12-11-15 11:10:09 UTC

Page Contents

- Overview
- Creating and Editing Users
 - Administrator Users
 - The Anonymous User
- Creating and Editing Groups
 - Default Groups
 - Admin Privileges for a Group
- User Management
 - Setting Groups for a User
 - Setting Users for a Group
- Recreating the Default Admin User
 - Obtaining a Security Configuration File
 - Resetting the Admin Password
 - Replacing the Security Configuration File
 - User
 - Password
- Disabling Remember Me at Login

Create a new user by clicking **New** at the top of the users table.

Only administrators can create users

To create users you must be an administrator (unless you are using external authentication such as LDAP)

Add New User

User Settings

User Name *

Email Address *

Admin

Disable UI access

Can Update Profile

Disable Internal Password ?

Set Password

Password *

Retype Password *

Password Strength

Cancel Save

In the **New User** (or **Edit User**) dialog you can set the **User Name**, **Email Address** and **Password** for the user as well as the following parameters:

Admin	When set, this user is an administrator with all the ensuing privileges. For more details please refer to Administrator Users .
Disable UI Access	When set, this user can only access Artifactory through the REST API.
Can Update Profile	When set, this user can update his profile details (except for the password. Only an administrator can update the password). There may be cases in which you want to leave this unset to prevent users from updating their profile. For example, a departmental user with a single password shared between all department members.
Disable Internal Password	When set, disables the fallback of using an internal password when external authentication (such as LDAP) is enabled.

Artifactory stores passwords as hashes or encrypted hashes.

If the user has generated an API key, you can revoke it from the **Actions** menu.

Edit admin User

User Settings

User Name *

Email Address *

⌵ Actions

- ⓧ Revoke Api Key
- ⓧ Delete User

Administrator Users

An administrator user is to Artifactory as a "root" is to UNIX systems. Administrators are not subject to any security restrictions, and we therefore recommend to create a minimum number of administrators in your system.

You can control which permission-targets administrators have access to thereby assigning responsibility for a specific repository path. For details please refer to [Managing Permissions](#).

The Default Admin Account

The default user name and password for the built-in administrator user are: **admin/password**.

You should change the password after first log in. If you forget the admin account password, you can recover it. Please refer to [Recreating the Default Admin User](#).

The Anonymous User

Artifactory supports the concept of anonymous users and installs with a pre-defined `anonymous` user to which you can assign [permissions](#) just like for any other user.

Anonymous access can be controlled under [Security General Configuration](#). Set **Allow Anonymous Access** to activate the anonymous user. The anonymous user must be activated before you can fine tune its permissions.

When anonymous access is activated, anonymous requests can download cached artifacts and populate caches, regardless of other permissions defined.

Creating and Editing Groups

A group represents a role in Artifactory and is used with RBAC (Role-Based Access Control) rules.

To manage groups, in the **Admin** module select **Security | Groups**.

Groups Management

6 Groups ⊕ New

Filter by Group Name

⊗ Delete < Page 1 of 1 >

<input checked="" type="checkbox"/>	Group Name ▲	Permissions	External	Admin	Auto Join
<input checked="" type="checkbox"/>	deployers	-			
<input checked="" type="checkbox"/>	hyperv-administrators	-		<input checked="" type="checkbox"/>	
<input checked="" type="checkbox"/>	jfrog-users	-			
<input checked="" type="checkbox"/>	managers-il	-		<input checked="" type="checkbox"/>	
<input checked="" type="checkbox"/>	qa-il	-			
<input checked="" type="checkbox"/>	readers	1 Anything			<input checked="" type="checkbox"/>

Create a new group by clicking **New** at the top of the groups table.

Add New Group

Group Settings

Group Name * Description

Automatically Join New Users to this Group

Admin Privileges

You must assign a unique name to each group and can add an optional description

Default Groups

When creating (or editing) a group you can set **Automatically Join New Users to this Group**.

When this parameter is set, any new users defined in the system are automatically assigned to this group.

This is particularly useful if users are defined automatically and you want them to be assigned to certain groups. For example, when using external authentication such as LDAP, users are automatically created on successful login and you can use this parameter to assign these users to particular groups by default.

Admin Privileges for a Group

If **Admin Privileges** is set, any users added to this group will automatically be assigned with admin privileges in the system. For reasons of security when Admin Privileges is set, **Automatically Join New Users to this Group** is disabled so that new users are not automatically provided with admin privileges.

User Management

There are two ways to manage users' assignment to groups:

1. [Setting the groups for a user](#)
2. [Setting the users for a group](#)

Setting permissions

In both cases, you can assign corresponding permissions to the user or group respectively on the same screen. For more details please refer to [Managing Permissions](#).

Setting Groups for a User

You can assign and remove a user from groups when the user is created or by editing user's details later.

In the **Admin** module, under **Security | Users**, from the list of users, select the user you wish to assign to or remove from groups.

In the **Related Groups** section of the form, you can set which groups the user should be assigned to.

Edit admins Group

Group Settings

Group Name *

admins

Description

admins group

Automatically join New Users to this Group

Users

Filter...

manager

test

yossis

roy

danny

ofer

dima

Filter...

yinon

matank

admin



Setting Users for a Group

You can assign and remove a users from a group by editing the group's details.

In the **Admin** module, under **Security | Groups**, from the list of groups, select the group you wish modify.

In the **Users** section of the form, you can set which users should be assigned to the group.

Edit admins Group

Group Settings

Group Name *

admins

Description

admins group

Automatically join New Users to this Group

Users

Filter...

manager

test

yossis

roy

danny

ofer

dima



Filter...

yinon



matank



admin



Recreating the Default Admin User

If you are unable to obtain administrator access, you will need to recreate the default administrator user in order to be able to manage users of your system using the following steps::

1. Obtain a security configuration file
2. Reset the admin password
3. Correctly place the security configuration file

Obtaining a Security Configuration File

The security configuration file is called `security.xml`.

If your instance of Artifactory is configured to perform [backups](#) automatically, you can find it in the root backup folder.

If Artifactory is **not** configured to perform backups automatically you need to force creation of a new `security.xml` file as follows:

- Remove the file `$ARTIFACTORY_HOME/data/.deleteForSecurityMarker` and restart Artifactory .
- Make sure that Artifactory completes the startup sequence without interruption
- The security configuration file with the current time stamp can be found in `$ARTIFACTORY_HOME/etc/security.<time stamp>.xml`

Resetting the Admin Password

Reset the admin password as follows:

- Make a copy of the `security.xml` file you obtained in the previous section
- In the copy, edit the admin's password field and enter the password hash code (according to your version of Artifactory) as follows:

Admin password hash code

For version 3.x and above:

```
<password>1f70548d73baca61aab8660733c7de81</password>
```

For version 2.x: `<password>5f4dcc3b5aa765d61d8327deb882cf99</password>`

Replacing the Security Configuration File

- Place the modified security configuration file under `$ARTIFACTORY_HOME/etc`
- Rename the file to `security.import.xml`
- Restart Artifactory

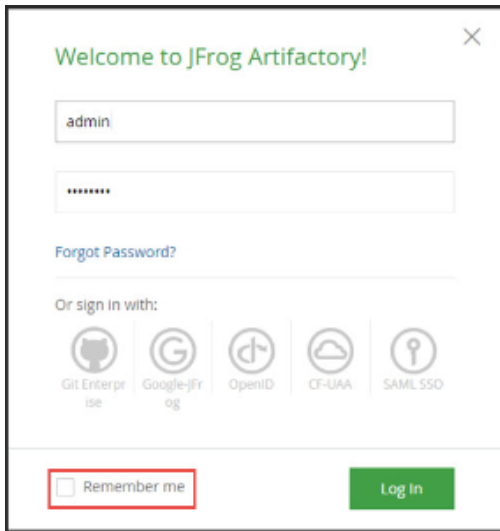
Once Artifactory has completed its startup sequence you will be able to login using the default admin user credentials:

User	admin
Password	password

Disabling Remember Me at Login

The Artifactory login screen includes a **Remember Me** checkbox. If the user sets this checkbox when logging in, Artifactory will store a cookie in the browser for a period of 7 days allowing the user to be logged in automatically when starting up Artifactory.

Once the cookie expires, the user will have to log in again.



An Artifactory administrator can disable this feature and force all users to enter their credentials at every login. To do so simply add the following property to `$ARTIFACTORY_HOME/etc/artifactory.system.properties` and restart Artifactory:

```
artifactory.security.disableRememberMe=true
```

Managing Permissions

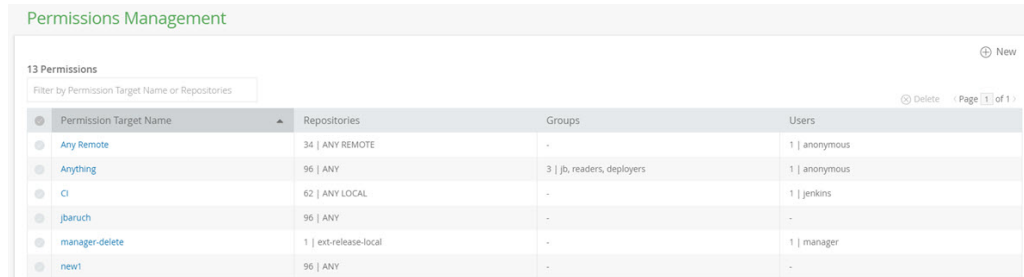
Overview

Artifactory allows you to control access to repositories via **Permission Targets**.

A permission target is comprised of a set of physical repositories (i.e. local or remote repositories - but not virtual ones), and a set of users or groups with a corresponding set of permissions defining how they can access the specified repositories. Include and Exclude patterns give you finer control over access to a specific set of artifacts within the repositories of the permission target.

For example, you can create a permission target that allows user "Builder" and group "Deployers" to read from and deploy artifacts to the `libs-releases` repository. Using the Include Pattern and Exclude Pattern settings you could implement finer control over specific artifacts within that repository if so desired.

To manage permissions, in the **Admin** module go to **Security | Permissions**.



The screenshot shows the 'Permissions Management' interface. At the top, there is a 'New' button and a search filter 'Filter by Permission Target Name or Repositories'. Below the filter is a table with 13 rows. The table has four columns: 'Permission Target Name', 'Repositories', 'Groups', and 'Users'. The rows are as follows:

Permission Target Name	Repositories	Groups	Users
Any Remote	34 ANY REMOTE	-	1 anonymous
Anything	96 ANY	3 jenkins, readers, deployers	1 anonymous
CI	62 ANY LOCAL	-	1 jenkins
jbaruch	96 ANY	-	-
manager-delete	1 ext-release-local	-	1 manager
new1	96 ANY	-	-

Page Contents

- Overview
- Creating a Permission Target
 - Permission Target Managers
 - Preventing Overwriting Deployments
- Examining Permissions
 - By Repository
 - By User or Group

Creating a Permission Target

To create a **Permission Target**, in the **Permissions Management** page click "New" to display the **New Permission** screen.

New Permission

Name *

Sample Permission

Repositories Groups Users

Include Pattern ⓘ

New Pattern

Exclude Pattern ⓘ

New Pattern

**

Filter...

Filter...

Available Repositories

Selected Repositories

- libs-snapshot-local X
- nuget-local X
- bower-remote X
- chef-remote X
- cocoapods-remote X
- debian-remote X
- jcenter X
- nuget-remote X

Any Local Repository Any Remote Repository Any Distribution Repository

Cancel Save & Finish

Name

You must provide a unique name for each **Permission Target** (limited to 64 characters).

Repositories

Select the repositories to which this **Permission Target** applies. You can use the **Any Local Repository** or **Any Remote Repository** check boxes as a convenience.

Include and Exclude Patterns

Using an "Ant-like" expressions, you can specify any number of Include or Exclude Patterns as a comma-separated list in the corresponding entry field (limited to 1024 characters in total).

In the example above, source files have been excluded from the **Permission Target** named "Not sources" using the appropriate **Exclude Pattern**.

User and Group Permissions

Using the corresponding tabs, you can set the permissions granted to a user or a group. Double-click the user or group you want to modify to add it to the list of **Principals**, and then check the permissions you wish to grant.

You cannot add a user or group with admin privileges to a Permission Target

Since an admin is privileged with all permissions, you cannot add a user or group with admin privileges to a Permission Target.

New Permission

Name *

Sample Permission

Repositories | **Groups** | Users

Filter...

- readers >
- jfrog-users >
- hyperv-administrators >
- managers-il >

2 Records

Filter by Group

Remove < Page 1 of 1 >

<input checked="" type="checkbox"/>	Group	Manag...	Delete/Overwri...	Deploy/Cache	Annota...	Read
<input checked="" type="checkbox"/>	deployers	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	qa-il	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Double click to add group

Cancel Save & Finish

The available permissions are as follows:

Manage	Allows changing the permission settings for other users on this permission target
Delete/Overwrite	Allows deletion or overwriting of artifacts
Deploy/Cache	Allows deploying artifacts and deploying to caches (i.e. populating caches with remote artifacts)
Annotate	Allows annotating artifacts and folders with metadata and properties
Read	Allows reading and downloading of artifacts

Multiple Permissions

Permissions are additive and must be explicitly granted. If a checkbox is not set for a user, then that user does not have the corresponding permission.

Permission Target Managers

By assigning the **Manage** permission to a user, you may designate them as the "Permission Target Manager". These users may assign and modify permissions granted to other users and groups for this **Permission Target**. In the Artifactory UI these users have access to the specific users they are allowed to manage. This can be useful on a multi-team site since you can delegate the responsibility of managing specific repositories to different team members.

Preventing Overwriting Deployments

You can prevent a user or group from overwriting a deployed release or unique snapshot by not granting the **Delete** permission. Non-unique snapshots can always be overwritten (provided the **Deploy** permission is granted).

Examining Permissions

You can examine permissions in the context of repositories, users or groups.

By Repository

In the **Artifacts** module, select repository you want to view in the **Artifact Repository Browser** and then select the **Effective Permissions** tab to see the permissions granted to users or groups for this repository.

The screenshot shows the 'Artifact Repository Browser' interface. On the left is a tree view of repositories, with 'chef-local' selected. The main panel shows the 'Effective Permissions' tab for 'chef-local'. It displays a list of 7 users with their respective permissions across various actions like Delete/Overwrite, Deploy/Cache, Annotate, and Read.

Principal	Permission Targets	Delete/Ov...	Deploy/...	Ann...	Re...
admin	-	✓	✓	✓	✓
anonymous	2 Any Remote, Anything		✓		✓
franku	1 Anything	✓	✓	✓	✓
janep	1 Anything	✓	✓	✓	✓
johnk	1 Anything	✓	✓	✓	✓
ramih	1 Anything	✓	✓	✓	✓
yuvalr	1 Anything				✓

By User or Group

For any user or Group, you can view the list of Permission Targets that it is associated with (whether directly or through membership in a group).

For users, In the **Admin** module, under **Security | Users**, select the user you wish to examine. The **User Permissions** are displayed at the bottom of the user's page.

The screenshot shows the 'User Permissions' section for a user. It displays a list of 2 permissions with their associated targets and repositories.

Permission Target	Applied To	Repositories	Manage	Delete/Overwrite	Deploy/Cache	Annotate	Read
Anything	readers	16 ANY					✓
Example Permission	yuvalr	3 debian-local, libs-s...		✓	✓	✓	✓

You can similarly view Group permissions in the **Admin** module under **Security | Groups**.

Centrally Secure Passwords

Overview

Some tools use cleartext passwords, which can pose a security risk. The security risk is even greater if you use LDAP or other external authentication, since you expose your SSO password in cleartext and that password is likely to be used for other services, not just Artifactory.

For example, Maven uses cleartext passwords in the `settings.xml` file by default.

Using Maven's built-in support for encrypted passwords and generating passwords on the client side does not overcome the security risks for the following reasons:

1. The login password is decrypted on the client side and ends up as cleartext in memory, and then transmitted over the wire (unless forcing SSL too).
2. The master password used for decryption is stored in clear text on the file system.
3. Password encryption is left to the good will of the end-user and there is no way to centrally mandate it.

Artifactory provides a unique solution to this problem by generating encrypted passwords for users based on secret keys stored in Artifactory. You can ensure users' shared passwords are never stored or transmitted as clear text.

Page Contents

- [Overview](#)
- [Using Your Secure Password](#)

You can set a central policy for using or accepting encrypted passwords in the **Admin** module under **Security | General** by setting the **Password Encryption Policy** field.

Security General Configuration

Allow Anonymous Access
 Prevent Anonymous Access to Build Related Info

Hide Existence of Unauthorized Resources

 Password Encryption Policy

SUPPORTED

The behavior according to the **Password Encryption Policy** setting is as follows:

<i>Supported</i>	Artifactory can receive requests with encrypted password (default).
<i>Required</i>	Artifactory requires an encrypted password for every authenticated request.
<i>Unsupported</i>	Artifactory will reject requests with encrypted password.

Using Your Secure Password

To secure your password:

1. Open your profile page (click on your login name on the upper-right corner), type-in your password in the **Current Password** field and click **Unlock**.

User Profile: admin

Current Password

Unlock

🔑 Insert the password and press the Unlock button to edit the profile.

2. Once your profile is unlocked, click the corresponding icons next to your encrypted password to view it openly or copy it to the clipboard.

User Profile: admin

Personal Settings

New Password

Retype Password

Email Address *

Password Strength

Authentication Settings

API Key ?

⊗ Revoke API Key

Encrypted Password

Different encryption mechanisms

The encryption mechanisms of the Oracle and IBM JDKs are not identical. Switching from one to another will make your encrypted password obsolete

IBM JDK Encryption Restrictions

Some of the IBM JRE/JDK are shipped with a restriction on the encryption key size (mostly for countries outside the US); This restriction can be officially removed by downloading unrestricted policy files from IBM and overriding the existing ones:

1. Register and download the unrestricted JCE policy files from the [IBM website](#).
2. Select the correct zip that matches your JAVA version.
3. The downloaded zip file contains 2 jar files - *local_policy.jar* and *US_export_policy.jar*. Backup the existing files in `$IBM_JDK_HOME/jre/lib/security` and extract the jars from the zip file to this location
4. Restart Artifactory

Master Key Encryption

Overview

The [global Artifactory configuration file](#) stores the various passwords that are needed in order to interface with your organizations systems and external repositories. For example, Artifactory may need your LDAP server password.

In order to keep these passwords secure, you can choose to store them in an encrypted format. In this case, Artifactory will generate a **Master Encryption Key** which will be used to encrypt these passwords for storage and display, and to decrypt them when you need to access the corresponding resources.

IBM JDK Encryption Restrictions

Users of the IBM JDK should read about IBM JDK encryption restrictions described in [Using Your Secure Password](#).

Page Contents

- [Overview](#)
- [Encrypting Passwords](#)

- Decrypting Passwords
- Exporting and Importing the Master Key
 - Master Key File Location

Encrypting Passwords

When Master Key Encryption is activated all current passwords in the [global configuration file](#) are encrypted, and any new passwords, or updates will also be encrypted automatically.

By default Artifactory is configured to encrypt passwords. An Artifactory administrator can activate and deactivate encryption by either using the [REST API](#), or through the Artifactory UI in the **Admin** module under **Security | General**.

General Security Configuration

General Security Settings

- Allow Anonymous Access
- Prevent Anonymous Access to Build Related Info
- Hide Existence of Unauthorized Resources [?](#)

Password Encryption Policy [?](#)

Supported

Once Master Key Encryption is activated, subsequent activation using the REST API are ignored.

Decrypting Passwords

An Artifactory administrator can deactivate encryption, and decrypt any currently encrypted passwords by either using the [REST API](#), or through the Artifactory UI in the **Admin** module under **Security | General**.

When you select **Decrypt**, all passwords in the global configuration file are decrypted, the configuration is reloaded and the current Master Key is removed.

Any new passwords entered, or passwords updated will not be encrypted.

Exporting and Importing the Master Key

If the Master Key is in its default location under the `ARTIFACTORY_HOME/etc` folder, it will be exported during a [system backup](#) or [full system export](#).

Correspondingly, if a Master Key was exported, and you now perform a full system import, the key will be copied to the default location and the Master Key Encryption feature will be activated. i.e. the Master Key will be used to encrypt and decrypt the imported configuration.

Master Key File Location

By default, the Master Key file is located under `ARTIFACTORY_HOME/etc/security/artifactory.key`.

You may wish to exercise more stringent security so that the master key file is in a more secure location.

In this case you can change the file location by modifying the `artifactory.security.master.key` property in the `artifactory.system.properties` file.

For example,

Modifying the default master key file location

```
artifactory.security.master.key=<other location>/artifactory.key
```

If you use a partial path, then it will be interpreted as relative to the `$ARTIFACTORY_HOME/etc` folder.

If you change the Master Key file location, it will not be exported automatically. It is up to the administrator to back it up along with the export, and restore it manually on an import.

Managing Security with LDAP

Introduction

Artifactory supports authenticating users against an LDAP server out-of-the-box.

When LDAP authentication is active, Artifactory first attempts to authenticate the user against the LDAP server. If LDAP authentication fails, Artifactory tries to authenticate via its internal database.

For every LDAP authenticated user Artifactory creates a new user in the internal database (provided the user does not already exist), and automatically assigns that user to the default groups.

Managing Permissions for LDAP Groups

Artifactory can synchronize your LDAP groups and leverage your existing organizational structure when managing group-based permissions. LDAP groups in Artifactory use super-fast caching and support Static, Dynamic and Hierarchical mapping strategies.

Powerful management is accomplished with multiple, switchable LDAP settings and visual feedback about the up-to-date status of groups and users coming from LDAP. The LDAP Groups feature is bundled as one of the Add-ons included in Artifactory Pro.

For full details on how to synchronize your LDAP Groups with Artifactory, please refer to [LDAP Groups](#).

Using Active Directory?

If you are using Active Directory to authenticate users, please refer to [Managing Security with Active Directory](#).

Page Contents

- [Introduction](#)
- [Configuration](#)
 - [Non-UI Authentication Cache](#)
- [Avoiding Clear Text Passwords](#)
- [Preventing Authentication Fallback to the Local Artifactory Realm](#)
- [Using LDAPS \(Secure LDAP\)](#)
- [Watch the Screencast](#)

Configuration

To configure LDAP authentication, in the **Admin** module go to **Security | LDAP** and click **New**.

New LDAP Settings

LDAP Settings

Enabled

Settings Name *

LDAP URL * [?](#)

Auto Create Artifactory Users [?](#)

Allow Created Users Access To Profile Page [?](#)

User DN Pattern [?](#)

Email Attribute [?](#)

Search Filter [?](#)

Search Base [?](#)

Secure LDAP Search
Protect against LDAP poisoning by filtering out users exposed to vulnerability.

Search Sub-tree [?](#)

Manager DN [?](#)

Manager Password [?](#)

Test LDAP Connection

Test User Name Test Password

The configuration parameters for LDAP connection settings are as follows:

Settings Name	The unique ID of the LDAP setting.
Enabled	When set, these settings are enabled.

LDAP URL	Location of the LDAP server in the following format: <code>ldap://myserver:myport/dc=sampledomain,dc=com</code> . The URL should include the base DN used to search for and/or authenticate users.
Auto Create Artifactory Users	When set, Artifactory will automatically create new users for those who have logged in using LDAP, and assign them to the default groups.
Allow Created Users Access To Profile Page	When set, users created after logging in using LDAP will be able to access their profile page in Artifactory.
User DN Pattern	A DN pattern used to log users directly in to the LDAP database. This pattern is used to create a DN string for "direct" user authentication, and is relative to the base DN in the LDAP URL. The pattern argument {0} is replaced with the username at runtime. This only works if anonymous binding is allowed and a direct user DN can be used (which is not the default case for Active Directory). For example: <code>uid={0},ou=People</code>
Email Attribute	An attribute that can be used to map a user's email to a user created automatically by Artifactory.
Search Filter	A filter expression used to search for the user DN that is used in LDAP authentication. This is an LDAP search filter (as defined in 'RFC 2254') with optional arguments. In this case, the <code>username</code> is the only argument, denoted by ' <code>{0}</code> '. Possible examples are: <code>uid={0}</code> - this would search for a username match on the uid attribute. Authentication using LDAP is performed from the DN found if successful.
Search Base	The Context name in which to search relative to the base DN in the LDAP URL. Multiple search bases may be specified separated by a pipe (). This is parameter is optional.
Manager DN	The full DN of a user with permissions that allow querying the LDAP server. When working with LDAP Groups, the user should have permissions for any extra group attributes such as memberOf .
Manager Password	The password of the user binding to the LDAP server when using "search" authentication.
Search Sub Tree	When set, enables deep search through the sub-tree of the LDAP URL + Search Base. True by default.

Non-UI Authentication Cache

You can configure Artifactory to cache data about authentication against external systems such as LDAP for REST API requests. This means that the first time a user needs to be authenticated, Artifactory will query the external system for the user's permissions, group settings etc.

The information received from the external system is cached for a period of time which you can configure in the `$ARTIFACTORY_HOME/etc/artifactory.system.properties` file by setting the `artifactory.security.authentication.cache.idleTimeSecs` property.

This means that once a user is authenticated, while the authentication data is cached, Artifactory will use the cached data rather than querying the external system, so authentication is much faster

By default this is set to 300sec.

REST API Only

The cache is only relevant for REST API requests, and is not relevant when using the Artifactory UI.

Avoiding Clear Text Passwords

Storing your LDAP password in clear text in `settings.xml` on your disk is a big security threat, since this password is very sensitive and is used

in SSO to other resources in the domain.

When using LDAP, we strongly recommend, using [Artifactory's Encrypted Passwords](#) in your local settings.

Preventing Authentication Fallback to the Local Artifactory Realm

In some cases, as an administrator you may want to require users to authenticate themselves through LDAP with their LDAP password. However, if a user already has an internal account with a password in Artifactory, Artifactory can fallback to use his internal password if LDAP authentication fails.

You can prevent this fallback authentication by ensuring that the **Disable Internal Password** checkbox in the [Edit User](#) dialog is set.

Using LDAPS (Secure LDAP)

To use LDAPS with a valid certificate from a CA trusted by Java, all you need to do us use a secure LDAP URL in your settings, e.g. `ldaps://secure_ldap_host:636/dc=sampledomain,dc=com`.

If you want to use LDAPS with a non-trusted (self-signed) certificate, please follow the steps described in [Using a Self-Signed Certificate](#).

Watch the Screencast

Managing Security with Active Directory

Introduction

Artifactory supports integration with an Active Directory server to authenticate users and synchronize groups.

When authentication using Active Directory is configured and active, Artifactory first attempts to authenticate the user against the Active Directory server. If the authentication fails, Artifactory tries to authenticate via its internal database.

For every externally authenticated user configured in your Active Directory server, Artifactory creates a new user in the internal database (provided the user does not already exist), and automatically assigns that user to the default groups.

Page Contents

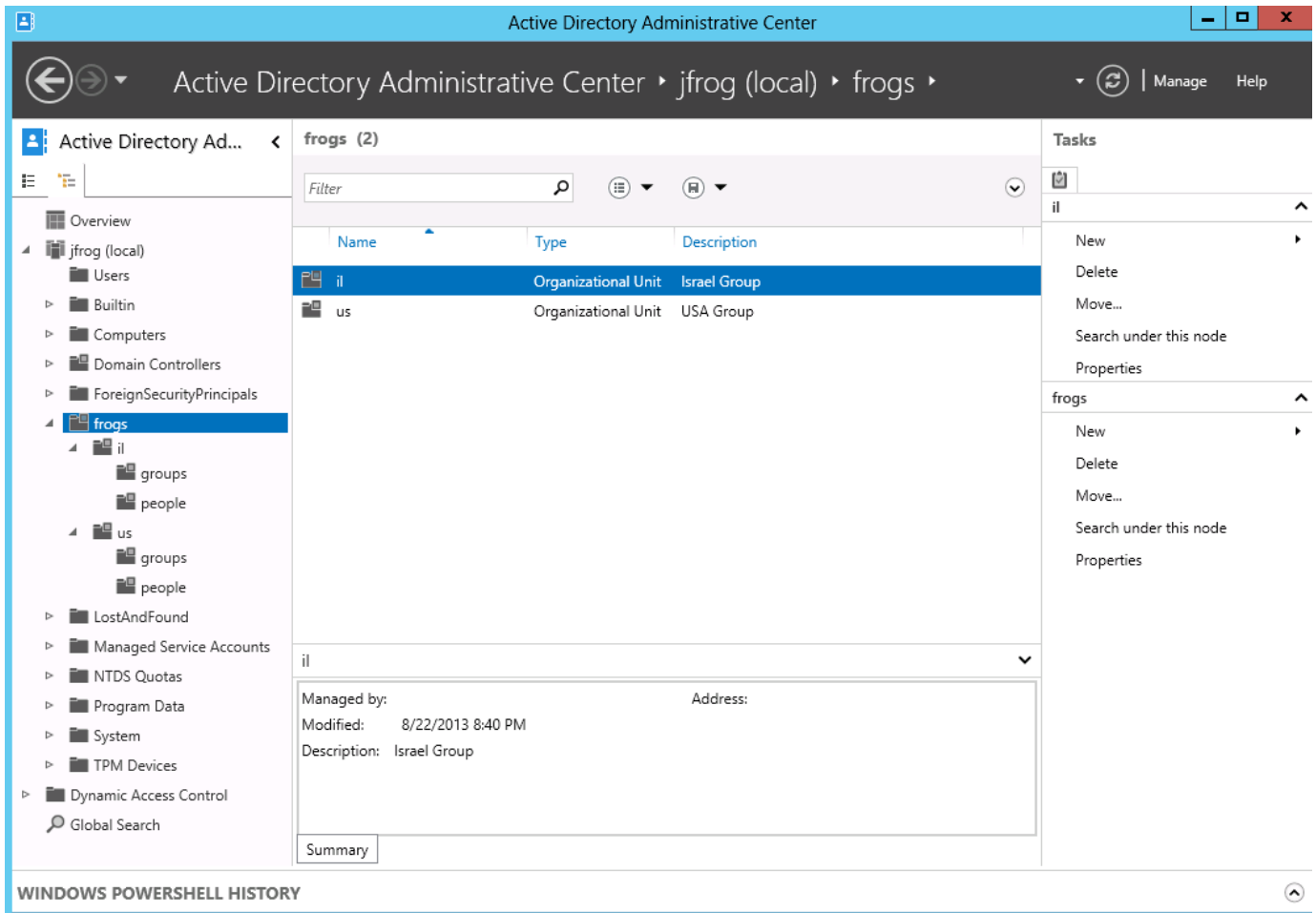
- [Introduction](#)
- [Working With Active Directory](#)
- [Importing Active Directory Groups](#)
 - [Support for Nested Groups](#)
- [Using Secure Active Directory](#)

Working With Active Directory

We will describe how to configure Artifactory to work with Active Directory using an example.

Consider an Active Directory server that must support the following conditions:

- Users are located in two geographically separated sites. Some are in the US (designated as "us"), while others are in Israel (designated as "il").
- Each site defines users and groups in different places in the Active Directory tree as displayed below.



To configure Active Directory authentication, in the **Admin** module, go to **Security | LDAP** and click **New**.

New LDAP Settings

LDAP Settings

Enabled

Settings Name *

frogs-ii

LDAP URL *

ldap://win2012.jfrog.local:389/dc=jfrog,dc=local

Auto Create Artifactory Users

User DN Pattern

Email Attribute

mail

Search Filter

sAMAccountName={0}

Search Base

ou=ii,ou=frogs | ou=us,ou=frogs

Manager DN

cn=Administrator,cn=Users,dc=jfrog,dc=local

Manager Password

.....

Sub-tree Search

Test LDAP Connection

Test User Name

Test Password

Test Connection

Cancel

Create

The configuration parameters are as follows:

Settings Name	The unique ID of the Active Directory setting.
Enabled	When set, these settings are enabled.
Active Directory URL	Location of the Active Directory server LDAP access point in the following format: <i>ldap://myserver:myport/dc=sampledomain,dc=com</i> . The URL may include the base DN used to search for and/or authenticate users. If not specified, the Search Base field is required.
User DN Pattern	A DN pattern used to log users directly in to the LDAP database. For Active Directory, we recommend leaving this field blank since this only works if anonymous binding is allowed and a direct user DN can be used, which is not the default case in Active Directory.

Auto Create Artifactory Users	When set, Artifactory will automatically create new users for those who have logged in using Active Directory. Any newly created users will be associated to the default groups.
Email Attribute	An attribute that can be used to map a user's email to a user created automatically by Artifactory. This corresponds to the mail field in Active Directory.
Search Filter	A filter expression used to search for the user DN that is used in Active Directory authentication. This is an LDAP search filter (as defined in 'RFC 2254') with optional arguments. In this case, the <code>username</code> is the only argument, denoted by <code>{0}</code> . For Active Directory the corresponding field should be <code>sAMAccountName={0}</code> .
Search Base	The Context name in which to search relative to the base DN in the Active Directory URL. This parameter is optional, but if possible, we highly recommend that you set it to prevent long searches on the Active Directory tree. Leaving this field blank will significantly slow down the Active Directory integration. The configuration in the example below indicates that search should only be performed under "frogs/il" or "frogs/us". This improves search performance since Artifactory will not search outside the scope of the "frogs" entry.
Manager DN	The full DN of a user with permissions that allow querying the Active Directory server. When working with LDAP Groups, the user should have permissions for any extra group attributes such as memberOf .
Manager Password	The password of the user binding to the Active Directory server when using "search" authentication.
Search Sub Tree	When set, enables deep search through the sub-tree of the Active Directory URL + Search Base. True by default.

Importing Active Directory Groups

Active Directory groups can be imported using either a **Static** mapping strategy or a **Dynamic** one (Active Directory works for both).

The only difference is in the attribute defined on the corresponding Active Directory entry:

- The Static mapping strategy defines a "member" multi-value attribute on the **group** entry containing user DNs of the group members
- The "Dynamic" configuration defines a "memberOf" multi-value attribute on the **user** entry containing group DNs of the groups the user is a member of.

Active Directory supports both configurations, so you can choose the one which fits your organization's structure.

New LDAP Group Setting

LDAP Group Settings

Settings Name *	LDAP Setting
<input type="text" value="allfrogs"/>	<input type="text" value="frogs"/>
Mapping Strategy: Static Dynamic Hierarchy	
Group Member Attribute *	Group Name Attribute *
<input type="text" value="member"/>	<input type="text" value="cn"/>
Description Attribute *	Filter *
<input type="text" value="description"/>	<input type="text" value="(objectClass=group)"/>
Search Base	<input checked="" type="checkbox"/> Sub-tree Search
<input type="text" value="ou=frogs"/>	

Synchronize LDAP Groups

< page 1 of 1 >

<input type="checkbox"/>	Group Name	Description	Sync ...
<input type="checkbox"/>	dev-il	R&D Israel	<input type="checkbox"/>
<input checked="" type="checkbox"/>	dev-us	R&D USA	

Cancel

Create

New LDAP Group Setting

LDAP Group Settings

Settings Name *	LDAP Setting
<input type="text" value="allfrogs"/>	<input type="text" value="frogs"/>
Mapping Strategy: <input type="radio"/> Static <input checked="" type="radio"/> Dynamic <input type="radio"/> Hierarchy	
Group Member Attribute *	Group Name Attribute *
<input type="text" value="memberOf"/>	<input type="text" value="cn"/>
Description Attribute *	Filter *
<input type="text" value="description"/>	<input type="text" value="(objectClass=group)"/>
Search Base	<input checked="" type="checkbox"/> Sub-tree Search
<input type="text" value="ou=frogs"/>	

Synchronize LDAP Groups

< page 1 of 1 >

<input type="checkbox"/>	Group Name	Description	Sync ...
<input type="checkbox"/>	dev-il	R&D Israel	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	dev-us	R&D USA	<input type="checkbox"/>

Cancel

Create

Support for Nested Groups

Artifactory supports synchronization with Active Directory "Nested Groups".

Microsoft provides a unique OID for rule chain matching as part of the [search filter syntax](#), so when executing an LDAP Query to Active Directory using this OID, Active Directory returns a list of all the groups that a user's main group membership inherits from.

The screenshot below shows the following example:

Mapping Strategy: Static

Group Membership Attribute: member:1.2.840.113556.1.4.1941:

Group Name Attribute: cn

Filter: (objectClass=group)

New LDAP Group Setting

LDAP Group Settings

Settings Name * ?

allfrogs

LDAP Setting ?

frogs

Mapping Strategy: **Static** | Dynamic | Hierarchy

Group Member Attribute * ?

member:1.2.840.113556.1.4.1941:

Group Name Attribute * ?

cn

Description Attribute * ?

description

Filter * ?

(objectClass=group)

Search Base ?

Sub-tree Search

Synchronize LDAP Groups

Search Group by Username (leave blank for *)



46 Records

Filter by Group Name

Import < Page 1 of 2 >

Group Name	Description	Sync S...
enterprise read-only domain con...	Members of this group are Read-Only Domain Controllers in the enterp...	<input checked="" type="checkbox"/>
winrmremotewmiusers_	Members of this group can access WMI resources over management pr...	<input checked="" type="checkbox"/>
cert publishers	Members of this group are permitted to publish certificates to the direct..	<input type="checkbox"/>

Cancel

Create

Using Secure Active Directory

To use Secure Active Directory with a valid certificate from a CA trusted by Java, all you need to do is use a secure Active Directory URL in your settings, e.g. `ldaps://secure_ldap_host:636/dc=sampldomain,dc=com`.

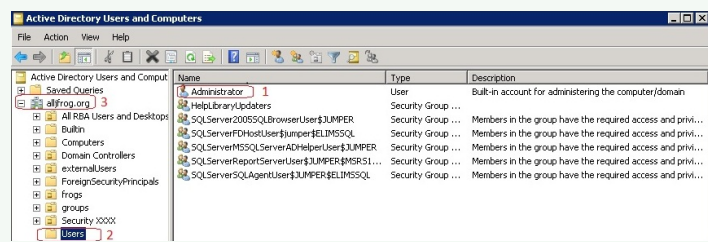
If you want to use Secure Active Directory with a non-trusted (self-signed) certificate, please follow the steps described in [Using a Self-Signed Certificate](#).

Manager DN

To construct the Manager DN string according to your Active Directory server, navigate to a user with administrator privileges (e.g. Administrator (1)), and then construct the Manager DN in reverse order (2,3) from the User, up the folder hierarchy.

For example, in this simple configuration, the Manager DN here should be
cn=Administrator,cn=Users,dc=alljfrog,dc=org

Notice that the domain (3) is split in reverse order to
dc=alljfrog,dc=org



Managing Certificates

Overview

Some remote repositories (e.g. Red Hat Networks) block access from clients that are not authenticated with an SSL/TLS certificate. Therefore, to use a remote repository to proxy such resources, Artifactory must be equipped with the corresponding SSL/TLS certificate.

To support this requirement when needed, from version 5.4, Artifactory lets you manage certificates and configure them for use by remote repositories.

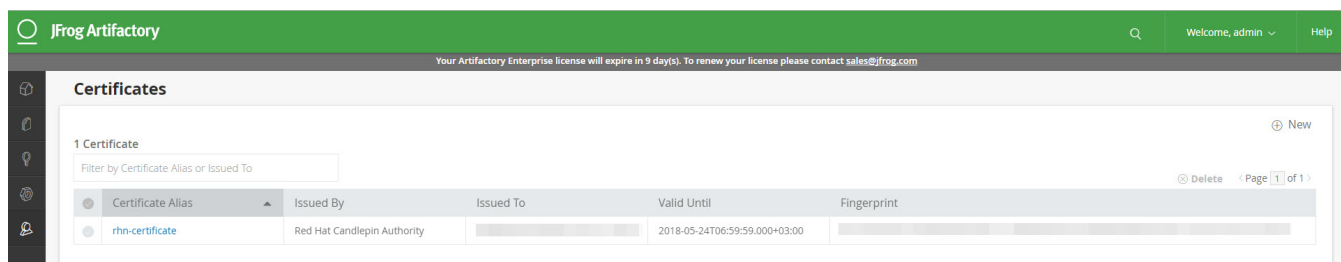
Page Contents

- Overview
- Adding Certificates
- Using a Certificate with a Remote Repository
- Proxying a Resource that Uses a Self-Signed Certificates
- REST API
 - Get Certificates
 - Add Certificate
 - Delete Certificate

Adding Certificates

Certificates are managed in the **Admin** module under **Security | Certificates**.


A certificate entered into this module should be a **PEM** file that includes both a private key and its corresponding certificate.



To add a new certificate, click **New**.

Add New Certificate ✕

Certificate Alias *



Copy your certificate or drop a .pem file

Provide the **Certificate Alias** and copy the certificate contents into the designated area. Alternatively, you can drag and drop the corresponding PEM file into the designated area.

To avoid text errors, we recommend dragging and dropping the PEM file into the designated area

Password-protected PEM files are not supported
Make sure the PEM file you upload is not password-protected.

Using a Certificate with a Remote Repository

When a remote repository proxy's a resource that requires authentication with a certificate, you need to obtain the certificate from the resource's owner and add it to the list of certificates as described above.

Under the remote repository's [Other Settings](#), select the certificate you want to use from the list provided in the **SSL/TLS Certificate** field.

Others

- Blacked Out ?
- Allow Content Browsing ?
- Store Artifacts Locally ?
- Synchronize Properties ?
- Block Mismatching Mime Types ?

SSL / TLS Certificate

|

rhn-certificate

New Mime Type +

Proxying a Resource that Uses a Self-Signed Certificates

If the remote resource that your Artifactory remote repository is proxying (e.g. Red Hat Network's server) uses an **untrusted** server certificate (i.e. it is **self-signed** and not signed by any known Certificate Authority), you need to import the server's certificate into Artifactory's JVM truststore. To learn more about configuring a Self-Signed Certificate in Artifactory, please refer to [Using a Self-Signed Certificate](#).

You cannot configure a self-signed certificate in Artifactory SaaS

If you are using Artifactory SaaS (as opposed to an on-prem installation), you will not be able to proxy resources that use untrusted (i.e. self-signed) certificates since you do not have access to the Artifactory SaaS JVM truststore.

REST API

Artifactory supports automated management of certificates using the REST API endpoints described below

Get Certificates

Gets a list of installed SSL certificates.

For details, refer to the REST API documentation for [Get Certificates](#).

Add Certificate

Installs a new SSL certificate.

For details, refer to the REST API documentation for [Add Certificate](#).

Delete Certificate

Deletes the specified certificate.

For details, refer to the REST API documentation for [Delete Certificate](#).

Using a Self-Signed Certificate

Overview

For several security features that you want to use over a secure connection (such as LDAPS, Secure Active Directory, or Secure OAuth), you may configure Artifactory to allow a non-trusted self-signed certificate

Page Contents

- [Overview](#)
- [Configuring a Self-Signed Certificate](#)

Configuring a Self-Signed Certificate

For outbound Artifactory connections (remote repositories, external authentication servers...) intended for SSL self-signed/internal CA signed certificates URL endpoints, you may use one of the following ways to establish trusts based on **your** certificates:

- Use the [instructions described on Oracle's documentation](#) to import a single/chain of certificates to your JVM's keystore.
 - Point Artifactory to use a custom certificate store. Follow the steps below (thanks to Marc Schoechlin for providing this information):
1. Download/acquire the certificate(s) of the SSL secured server `openssl s_client -connect <secure authentication server IP and port> -showcerts < /dev/null > server.ca`

Examples

LDAP or Active Directory:

```
openssl s_client -connect the.ldap.server.net:636 -showcerts < /dev/null > server.ca
```

OAuth (Use the Authorization URL). For example, with GitHub:

```
openssl s_client -connect github.com:443/login/oauth/authorize -showcerts < /dev/null > server.ca
```

2. Identify the CA certificate and keep only the ascii-text between BEGIN/END CERTIFICATE maker

3. Identify the standard `cacerts` file of your Java installation
4. Create a custom `cacerts` file by copying the `cacerts` file to the Artifactory configuration dir, e.g.


```
cp /usr/lib64/jvm/java-1_6_0-ibm-1.6.0/jre/lib/security/cacerts /etc/opt/jfrog/artifactory/
```
5. Import the CA certificate into the **customized** `cacerts` file


```
keytool -import -alias myca -keystore /etc/opt/jfrog/artifactory/cacerts -trustcacerts -file server.ca
=> Password: changeit
=> Agree to add the certificate
```
6. Change permissions for the artifactory user


```
chmod 755 /etc/opt/jfrog/artifactory/cacerts
chown artifactory:users /etc/opt/jfrog/artifactory/cacerts
```
7. Modify the defaults of the Artifactory JVM to use the custom `cacerts` file


```
echo "export JAVA_OPTIONS=\"\$JAVA_OPTIONS
-Djavax.net.ssl.trustStore=/etc/opt/jfrog/artifactory/cacerts\" " >>
/etc/opt/jfrog/artifactory/default
```
8. Restart Artifactory

Access Tokens

Overview

From version 5.0, Artifactory offers access tokens as a new and flexible means of authentication with a range of capabilities previously unavailable:

- **Cross-instance authentication**
Access tokens can be used for authentication, not only by the Artifactory instance or cluster where they were created, but also for other instances and clusters that are all part of the same "circle of trust" (described below).
- **User and non-user authentication**
The case for authenticating Artifactory users is clear, however access tokens can also be assigned to non-user entities such as CI server jobs.
- **Time-based access control**
Access tokens have an expiry period so you can control the period of time for which you grant access. However, you may also delegate that control to the receiving user by making them refreshable
- **Flexible scope**
By assigning Groups to tokens, you can control the level of access they provide.

To support these capabilities, an access token has the following properties:

Subject	The user to which this access token is associated. If the user specified does not exist, Artifactory will create a corresponding transient user. Artifactory administrators can assign a token to any subject (user); non-admin users who create tokens can only assign tokens to themselves.
Issuer	An identifier of the cluster on which the access token was created
Scope	The scope of access that the token provides. Access to the REST API is always provided by default; in addition, you may specify the group memberships that the token provides. Artifactory administrators can set any scope; non-admin users can only set the scope to a subset of the groups to which they belong.

Page Contents

- Overview
 - Access Service
 - Access Service Logs
 - Configuring Logging
- Cross-Instance Authentication
 - Setting the Private Key and Root Certificate
 - New Instances
 - Existing Instances
- Using Tokens
 - Basic Authentication
 - Authorization Headers
- Support Authentication for Non-Existing Users
- Generating Expirable Tokens
- Generating Refreshable Tokens
- Generating Admin Tokens
- Revoking Tokens
- REST API
 - Create Token
 - Refresh Token
 - Revoke Token
 - Get Service ID
- UI
- Troubleshooting

Expiry	The period of time from creation after which the token will expire. Artifactory administrators can set any expiry period; non-admin users can not change the expiry period so tokens they create expire after the default period of 60 minutes.
Refreshable	Whether the token may be refreshed for continued use or not
Audience	The set of Artifactory instances or clusters on which the token may be used identified by their Service IDs. The Service ID is a unique, internally generated identifier of an Artifactory instance or cluster and is obtained through Get Service ID REST API endpoint .

Access tokens are fully managed through [REST API](#) as described below.

Access Service

From Artifactory version 5.4, access tokens are managed under a new service called Access which is implemented in a separate WAR file, *access.war*. This change has no impact on how access tokens are used, however, the Artifactory installation file structure now also includes the added WAR file under the *\$ARTIFACTORY_HOME/webapps* folder. Artifactory communicates with the Access service over HTTP and assumes it is running in the same Tomcat using the context path of "access".

The new implementation is backwards compatible to old tokens, so you can still use tokens generated by an older version to authenticate in the new version, however, you cannot use tokens generated by the new version to authenticate in an older version.

Breaking Change: Note that the change is not forwards compatible, so tokens created from version 5.4 and above cannot be used for authentication with versions previous to 5.4. This may impact a circle of trust in which some instances are running versions below 5.4 while others are running version 5.4 and above.

Access Service Logs

The Artifactory Access Service uses the [Logback Framework](#) to manage logging. Activity is logged according to type in three different log files which can be found under the *\$ARTIFACTORY_HOME/access/logs* folder.

The following log files are available:

access.log	This is the main Access service log file containing data on the Access server activity
request.log	The HTTP traffic information for requests coming in. Most of these are from Artifactory
audit.log	Auditing of the actions performed by the service. Currently only successful actions are recorded (e.g. token was created, token was refreshed or revoked)

Since the Access service runs under the same Tomcat as Artifactory, its logs (catalina.out, localhost etc.) contains entries for both Artifactory and Access.

Configuring Logging

Logging for the Access service is configured in the *\$ARTIFACTORY_HOME/access/etc/logback.xml* file.

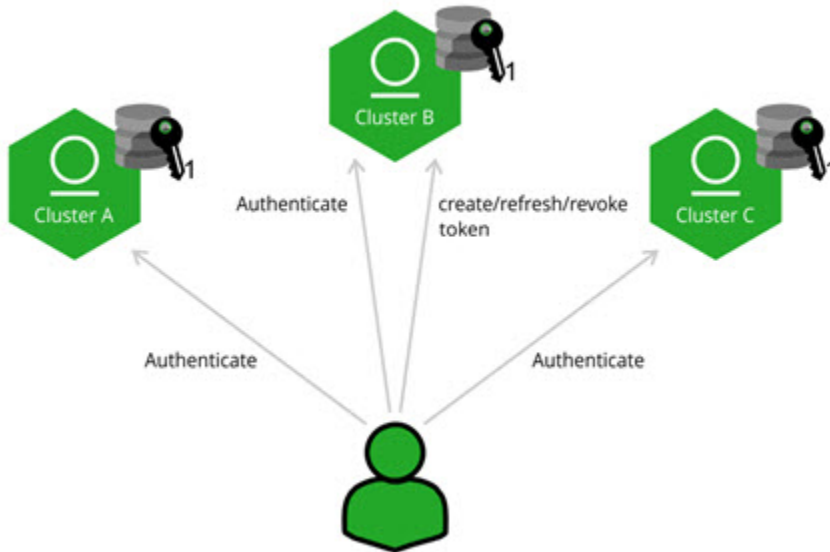
Cross-Instance Authentication

Access tokens support cross-instance authentication through a "circle of trust" established by sharing a private and public key pair among all participating instances. It is up to the Artifactory administrator to make sure that all participating instances are equipped with the same key pair. This means that any instance can generate a token to be used with any other instance within the circle of trust. When an Artifactory instance receives a REST API call authenticated by a signed token, it will use the root certificate that includes the public key to verify that its issuer is in the circle of trust.

Limitations

Only a token that is expirable and refreshable can be used for authentication on a different instance from the one that created it.

Only the issuing instance can refresh a token.



Setting the Private Key and Root Certificate

As mentioned above, it is up to the Artifactory administrator to make sure that all participating instances are equipped with the same key pair. The process to ensure this varies depending on whether you are bootstrapping new instances or setting up cross-instance authentication for existing instances.

New Instances

Artifactory Pro or OSS

1. Start up the first Artifactory instance (or cluster node for an HA installation) that will be in your circle of trust. A private key and root certificate are generated and stored under `$ARTIFACTORY_HOME/access/etc/keys`.
2. Copy the private key and root certificate files to a location on your file system that is accessible by all other instances/nodes that are in your circle of trust.
3. Before bootstrapping, for each of the other instances/nodes, create the `$ARTIFACTORY_HOME/access/etc` folder and create a properties file in it called `access.bootstrap.config` with the following contents:

```
key=/path/to/private.key
crt=/path/to/root.crt
```

4. When each instance/node starts up, if the `$ARTIFACTORY_HOME/access/etc/access.bootstrap.config` file exists, then the private key and root certificate are copied from the specified location into the server's home directory under `$ARTIFACTORY_HOME/access/etc/keys`.

Artifactory HA

In the case of an Artifactory HA installation, the private key and root certificate are included in the `bootstrap bundle`.

Existing Instances

Key rotation will invalidate any issued access tokens

The procedure below will create new key pairs which in turn will invalidate any existing Access Tokens issued by the current instance.

1. Copy the private key and root certificate files from the Artifactory instance whose circle of trust you want the current instance to join, to a location on your file system that is accessible by the current instance.
2. Before bootstrapping the instance:
 - a. Delete the existing private key and root certificate files (`private.key` and `root.crt`) from the `$ARTIFACTORY_HOME/access/etc` folder.
 - b. Create the `$ARTIFACTORY_HOME/access/etc/access.bootstrap.config` with the following contents:

```
key=/path/to/private.key
crt=/path/to/root.crt
```

- c. Add the following JVM property (under the `JAVA_OPTIONS` environment variable) to `$ARTIFACTORY_HOME/bin/artifactory.default`:

```
-Djfrog.access.force.replace.existing.root.keys=true
```

- d. Start up the instance ready to be added to your circle of trust and verify that the `artifactory.log` file shows the following entry:

```
*****
****
*** Forcing replacement of the root private key and
certificate ***
*****
*****
```

- e. Delete the JVM property you added to `$ARTIFACTORY_HOME/bin/artifactory.default` in step c.

Using Tokens

There are several ways you can use access tokens for authentication.

Basic Authentication

An access token can be used instead of a password for basic authentication. This may be useful when you need a client (such as certain dependency managers) that only supports basic authentication to access Artifactory. In this case, it is important to access Artifactory using the same user name provided when creating the token (with `-d "username=<USERNAME>"`).

For example, to use an access token as a password to ping Artifactory you could use:

```
curl -u<USERNAME>:<TOKEN> http://ARTIFACTORY_URL/api/system/ping
```

Authorization Headers

An access token can be used as a bearer token in authorization headers. This is especially useful for authenticating CI servers with Artifactory instead of using credentials, since you don't need to have a user defined in Artifactory if the group provided in `-d "member-of-groups:<GROUP>"` is configured in that Artifactory instance. As a result, there is no need to manage fictitious users for your different automation tools that need access to Artifactory.

For example, to use an access token as a bearer token to ping Artifactory you could use:

```
curl -H"Authorization: Bearer <TOKEN>"
http://ARTIFACTORY_URL/api/system/ping
```

Support Authentication for Non-Existing Users

One of the big advantages of access tokens is the fact that you don't have to create a user in Artifactory to use them. When creating a token,

you can specify a user name that does not exist, and Artifactory will create a transient user that will only exist as long as the token is valid. This can be useful to in giving access to different tools such as a CI server coordinating a build without having to manage fake user accounts. This method is also more secure since you can assign a new token for each "job" that the external tool runs.

Artifactory Administrator Only

Note that this feature is only available for Artifactory administrator since non-admin users can only create tokens with themselves as the Subject.

Generating Expirable Tokens

You can limit the validity period of a token by setting the expiry time when generating a token. If set, the token will be valid until the expiration time will pass.

You can all set a token to be non-expirable by setting the expiry to zero, in which case it will valid indefinitely until actively revoked.

This value is set by using the "&expires_in=<VALUE_IN_SECONDS>" param when generating the token (see example in REST API section below). If not used the default value will be 3600 meaning your token will be valid for one hour.

Artifactory Administrator Only

Note that only an Artifactory administrator can change the validity period of a token to any value. Non-admin users, can only set the token validity period to a value that is equal or less than the default 3600 seconds.

Generating Refreshable Tokens

As mentioned above, you can limit the validity period of an token by setting its expiry time. To allow extending access privileges of a token once it has expired, you can provide a refresh token which will generate a new token with the same privileges as the original one. This takes token management out of the hands of its issuer and delegates it to the user who received the token.

Who can refresh?

Only the instance (or HA cluster) that issued a refreshable token can actually refresh it.

Limitation

An external user who has created a token will still be able to refresh it even if he has been removed from the external authentication server.

Generating Admin Tokens

In general, the scope for a token is defined by specifying the groups into which the token is included, however, an Artifactory administrator can also create a token with admin privileges. This can be useful for JFrog Mission Control and JFrog Xray since both of these complementary applications require admin permissions to work seamlessly with Artifactory. With this capability, when Mission Control or Xray connect to an instance of Artifactory, they can create an admin tokens and use that for authentication instead of using basic authentication with a username and password.

Revoking Tokens

Any refreshable or non-expirable token can be revoked but only by the instance (or cluster) that issued it. A token with an expiry specified will lapse automatically upon reaching its expiry period (but can also be actively revoked earlier). A token that is not expirable (`expires_in` parameter is set to 0) must be actively revoked to terminate its usage. As described above, to support cross-site authentication, a token must be both expirable and refreshable. Note that this kind of token cannot be revoked. The only way to terminate its usage is to revoke its refresh token, so its usage will be terminated next time its expiry period lapses.

"Revoking" a cross-instance authentication token

To terminate usage of a token used for cross-instance authentication, you need to revoke its refresh token.

REST API

All management of access tokens is done via REST API through the endpoints described below.

Create Token

Creates an access token.

For details, refer to the REST API documentation for [Create Token](#).

Refresh Token

Refresh an access token to extend its validity. If only the access token and the refresh token are provided (and no other parameters), this pair is used for authentication. If username or any other parameter is provided, then the request must be authenticated by a token that grants admin permissions.

For details, refer to the REST API documentation for [Refresh Token](#).

Revoke Token

Revoke an access token

For details, refer to the REST API documentation for [Revoke Token](#).

Get Service ID

Provides the service ID of an Artifactory instance or cluster

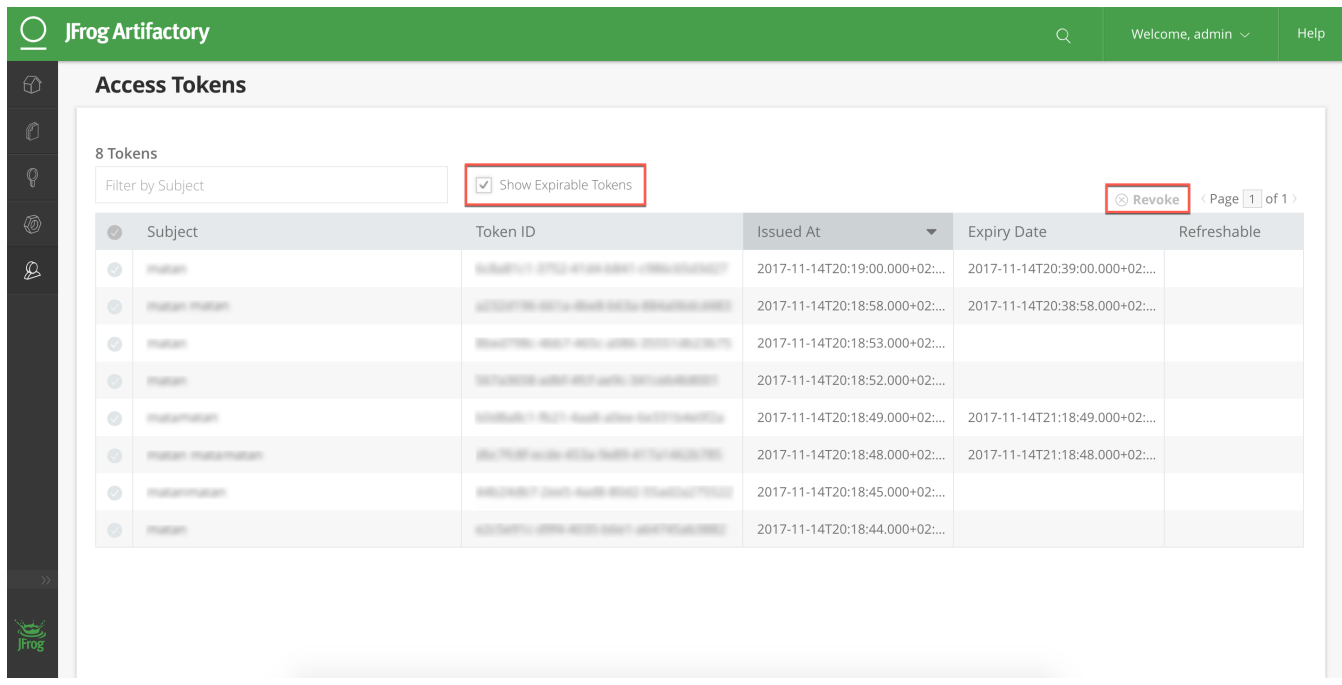
For details, refer to the REST API documentation for [Get Service ID](#).

UI

Admin users can view details on all created Access Tokens in the **Admin** module under **Security | Access Tokens**.

The Access Tokens page allows you to view, revoke, search by subject and filter to view only expirable tokens.

Additional functionalities, such as creating new tokens, is done via [REST API](#).



The screenshot shows the Jfrog Artifactory interface for managing Access Tokens. The page title is "Access Tokens" and it indicates there are 8 tokens. A search filter "Filter by Subject" is present, along with a checked checkbox for "Show Expirable Tokens". A "Revoke" button is highlighted in the top right of the table area. The table has the following columns: Subject, Token ID, Issued At, Expiry Date, and Refreshable. The table contains 8 rows of token data.

Subject	Token ID	Issued At	Expiry Date	Refreshable
...	...	2017-11-14T20:19:00.000+02:...	2017-11-14T20:39:00.000+02:...	
...	...	2017-11-14T20:18:58.000+02:...	2017-11-14T20:38:58.000+02:...	
...	...	2017-11-14T20:18:53.000+02:...		
...	...	2017-11-14T20:18:52.000+02:...		
...	...	2017-11-14T20:18:49.000+02:...	2017-11-14T21:18:49.000+02:...	
...	...	2017-11-14T20:18:48.000+02:...	2017-11-14T21:18:48.000+02:...	
...	...	2017-11-14T20:18:45.000+02:...		
...	...	2017-11-14T20:18:44.000+02:...		

Troubleshooting

- ▼ An exception is thrown for "java.lang.IllegalStateException: Provided private key and latest private key fingerprints mismatch"

Symptoms	During startup, Artifactory fails to start and an error is thrown: <i>java.lang.IllegalStateException: Provided private key and latest private key fingerprints mismatch.</i>
Cause	Artifactory tries to validate and compare access keys' fingerprint that reside on Artifactory's database and the local file system not match, the exception above will be thrown along with the mismatching fingerprint IDs. This could occur during an attempted upgrade/installation of Artifactory.
Resolution	Follow the steps below to make sure that all instances in your circle of trust have the same private key and root certificate: <div style="border: 1px solid red; padding: 5px; margin: 10px 0;"> <p>Key rotation will invalidate any issued access tokens The procedure below will create new key pairs which in turn will invalidate any existing Access Tokens.</p> </div> <p>a. Add the following JVM property (under the <code>JAVA_OPTIONS</code> environment variable) to <code>\$ARTIFACTORY_HOME/bin/default</code>:</p> <pre style="border: 1px dashed blue; padding: 10px; margin: 10px 0;">-Djfrog.access.force.replace.existing.root.keys=true</pre> <p>b. Start up the new instance and verify that the <code>\$ARTIFACTORY_HOME/logs/artifactory.log</code> or <code>\$ARTIFACTORY_HOME/access/logs/access.log</code> file shows the following entry:</p> <pre style="border: 1px dashed blue; padding: 10px; margin: 10px 0;">***** *** Forcing replacement of the root private key and certificate *****</pre> <p>c. Delete the JVM property you added to <code>\$ARTIFACTORY_HOME/bin/artifactory.default</code> in step a.</p>

▼ Following an upgrade of an Artifactory HA cluster node, the node fails to start up.

Symptoms	After correctly following the upgrade procedure, an Artifactory HA cluster node fails to start up
Cause	In Artifactory 5.4, the implementation of access tokens was taken out of the Artifactory WAR file and moved to a separate WAR file. As a result, your Tomcat's <code>server.xml</code> file needs to be modified.

Resolution Make sure that your \$ARTIFACTORY_HOME`tomcat/conf/server.xml` file is configured with 2 start/stop threads as shown in the example below (see `<Host name="localhost" appBase="webapps" startStopThreads="2"/>`):

```

<Server port="8015" shutdown="SHUTDOWN">
  <Service name="Catalina">

    <Connector port="8081" />

    <!-- This is the optional AJP connector -->
    <Connector port="8019" protocol="AJP/1.3" />

    <Engine name="Catalina" defaultHost="localhost">
      <Host name="localhost" appBase="webapps"
        startStopThreads="2" />
    </Engine>
  </Service>
</Server>

```

▼ The access token I generated is not working

Symptoms	Authentication with an access token doesn't work with an error that says "Token validation failed" .
Cause	The implementation of access tokens was changed in Artifactory 5.4. The change is backwards compatible, so tokens created with earlier versions of Artifactory can be authenticated in the new version, however the reverse is not true. Tokens created in versions 5.4 or later cannot be authenticated by versions earlier than 5.4.
Resolution	Either upgrade your older Artifactory instances, or make sure you only create access tokens with the older instances.

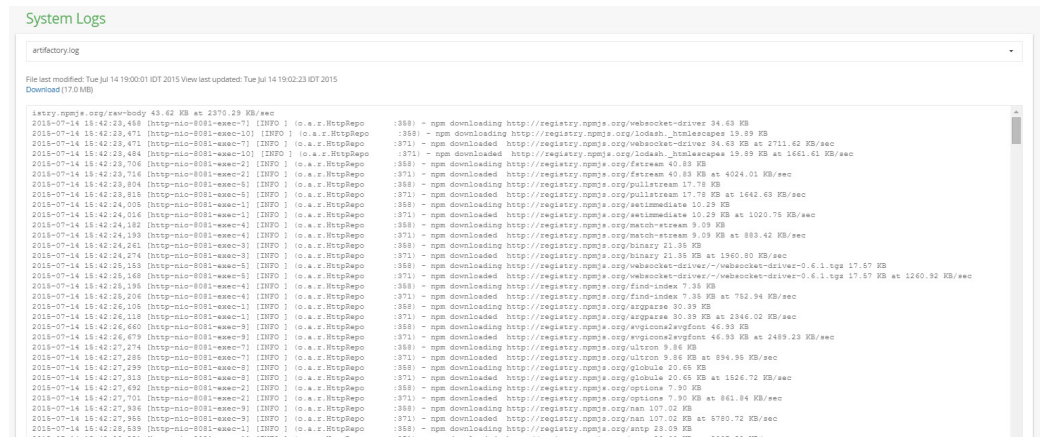
Access Log

The Artifactory access.log

Artifactory maintains an access log containing all security-related events, their source IP and context. Events include information on accept/reject of logins, and download, browsing and deployment of artifacts.

The access log is located at \$ARTIFACTORY_HOME/logs/access.log.

You can also view and download the access log from the Artifactory UI. In the **Admin** module go to **Advance d | System Logs**.



Page Contents

- [The Artifactory access.log](#)
- [Watches](#)

Watches

You can also choose to receive focused information about events for a specific repository section, using the [Watches Add-on](#).

Configuring a Reverse Proxy

Overview

In many cases, an organization may provide access to Artifactory through a reverse proxy such as [NGINX](#) or [Apache](#). In some cases, for example with Docker, this set up is even mandatory. To simplify configuring a reverse proxy, from version 4.3.1, Artifactory provides a **Reverse Proxy Configuration Generator** screen in which you can fill in a set of fields to generate the required configuration snippet which you can then download and install directly in the corresponding directory of your reverse proxy server. You can also use the [REST API](#) to manage reverse proxy configuration.

If you are using Artifactory behind a reverse proxy, we recommend that you set your [Custom URL Base](#) to match your [Artifactory Server Name](#).

Page Contents

- [Overview](#)
- [Reverse Proxy Settings](#)
- [Docker Reverse Proxy Settings](#)
 - [Using Subdomain](#)
 - [Using Port Bindings](#)
- [REST API](#)

Reverse Proxy Settings

To configure a reverse proxy, in the **Admin** module, select **Configuration | Reverse Proxy** and execute the following steps:

- Fill in the fields according to your configuration.
- Generate the configuration file. You may click the icons in the top right of the screen to view your configuration (which you may copy) or download it as a text file.
- Place the configuration file in the right place under your reverse proxy server installation and reload the configuration.

Using NGINX? Note these requirements.

To use NGINX as a reverse proxy to work with Docker, you need NGINX v1.3.9 or higher.

The NGINX configuration file should be placed under the `sites-enabled` directory.

For more details, please refer to [Configuring NGINX](#).

Using Apache? Note these requirements.

Some features in the Apache configuration are only supported from Apache HTTP Server v2.4.

To use Apache as your reverse proxy server, make sure you have the following modules installed and activated:

- proxy_http
- proxy_ajp
- rewrite
- deflate
- headers

- proxy_balancer
- proxy_connect
- proxy_html
- ssl
- lbmethod_byrequests
- slotmem_shm
- proxy

Support to generate Apache reverse proxy configuration is available from Artifactory version 4.4.1.

For more details, please refer to [Configuring Apache](#).

Best practice

When using a reverse proxy, we recommend passing it the **X-Artifactory-Override-Base-Url** header as follows:

For NGINX:

```
proxy_set_header X-Artifactory-Override-Base-Url $http_x_forwarded_proto://$<host>:<server
port>/<public context>
```

For Apache:

```
RewriteCond %{REQUEST_SCHEME} (.*)
RewriteRule (.*) - [E=my_scheme:%1]
[...]
RequestHeader set X-Artifactory-Override-Base-Url %{my_scheme}e://<server_name>/<app_context>
```


Reverse Proxy Settings

Web Server Type *

NGINX nginx

Internal Hostname * ?

localhost

Internal Port *

8080

Internal Context Path ?

artifactory

i Internal Artifactory URL: **localhost:8080/artifactory**

Public Server Name * ?

my-artifactory.org

Public Context Path ?

artifactory

Use HTTP

HTTP Port *

80

Use HTTPS

HTTPS Port *

443

SSL Key Path * ?

/etc/ssl/private/zmachine.io.key

SSL Certificate Path * ?

/etc/ssl/certs/zmachine.crt

i Users will have access to Artifactory at the following URL(s):

- **http://my-artifactory.org:80/artifactory**
- **https://my-artifactory.org:443/artifactory**

Web Server Type	The reverse proxy type.
Artifactory Server Name	The internal server name for Artifactory. If the Web Server is installed on the same machine as Artifactory you can use localhost , otherwise use the IP address or the machine name .
Artifactory Port	The port configured for Artifactory. The default value is 8081.

Artifactory Context Path	The path which will be used to access Artifactory. If Artifactory is accessible at the root of the server, leave this field empty.
Balance Members (Apache)	Only available in an Artifactory HA installation. Defines the group of servers in the HA cluster for load balancing. (default: artifactory). For more details, please refer to the NGINX documentation or Apache documentation accordingly.
Upstream Name (NGINX)	<div style="border: 1px solid #ccc; padding: 5px;"> <p>Multiple Artifactory instances under the same domain If using multiple Artifactory instances under the same domain, e.g. artdev.mycompany.org and artprod.mycompany.org you must assign a different names for balance members / upstream name to each cluster configuration since the session cookies will be available to both clusters and can cause an issue if trying to access both clusters in the same time.</p> </div>
Public Server Name	The server name which will be publicly used to access Artifactory within the organization.
Public Context Path	The path which will be publicly used to access Artifactory. If Artifactory is accessible on the root of the server leave this field empty.

You can configure access to Artifactory via HTTP, HTTPS or both (at least one is required). For each of these check boxes that you set, you need to fill in the corresponding fields as follows:

Use HTTP

HTTP Port *

80

Use HTTPS

HTTPS Port *

443

SSL Key Path * ? SSL Certificate Path * ?

/etc/ssl/private/myserver.key

/etc/ssl/certs/myserver.crt

i Users will access Artifactory in the following URL:

- <http://reverse-proxy:80/artifactory>
- <https://reverse-proxy:443/artifactory>

Use HTTP	When set, Artifactory will be accessible via HTTP at the corresponding port that is set.
HTTP Port	The port for access via HTTP. The default value is 80.
Use HTTPS	When set, Artifactory will be accessible via HTTPS at the corresponding port that is set.

HTTPS Port	The port for access via HTTPS. The default value is 443.
SSL Key Path	The full path to the key file for access via HTTPS.
SSL Certificate Path	The full path to the certificate file for access via HTTPS.

Docker Reverse Proxy Settings

When using Artifactory as a private Docker registry, the Docker client can only access Artifactory through a reverse proxy (Artifactory SaaS is an exception since it is external to your organization). Therefore, your Docker repositories must be configured with the corresponding Reverse Proxy settings in the **Docker Repository Configuration Advanced** tab. The **Reverse Proxy Configuration** screen also sets up your Docker Repository configuration.

There are two ways to configure Docker repositories to work with a reverse proxy: **Port** bindings or **Subdomain**.

Using Subdomain

If you select **Subdomain** as the **Reverse Proxy Method**, when configuring a Docker Repository, the **Registry Name** in the **Docker Repository Configuration Advanced** tab will be set automatically to the required value, and will use the **Repository Key** as the **Subdomain**.

Wildcard certificate

Using the **Subdomain** method requires a **Wildcard** certificate such as: ***.myservername.org**. You also need to ensure that the certificate you use supports the number of levels used in your subdomain.

Docker Reverse Proxy Settings in Reverse Proxy Configuration	Corresponding Reverse Proxy settings in Docker Repository Advanced Configuration
<p>Docker Reverse Proxy Settings</p> <p>Reverse Proxy Method: <input type="text" value="Sub Domain"/></p> <p>Server Name Expression: <input type="text" value="*.reverse-proxy"/></p> <p><small>When using sub domain method each Docker repository key will be used as the sub domain. You can view the repository URL in each Docker repository under the advanced tab. Example of docker push or pull command: <code>docker pull / push <REPOSITORY_KEY>.reverse-proxy/<IMAGE>:<TAG></code></small></p>	<p>Reverse Proxy</p> <p>Registry Name: <input type="text" value="docker-local.reverse-proxy"/></p> <p><small>To view / download the snippet, go to reverse proxy configuration page.</small></p>

Using Port Bindings

If you select **Port** as the **Reverse Proxy Method**, when configuring a Docker Repository, you will need to set the **Registry Port** in the **Docker Repository Configuration Advanced** tab. Together with the **Public Server Name**, this is the port the Docker client will use to pull images from and push images to the repository. Note that in order for all of your Docker repositories to be included in your reverse proxy configuration, you first you need to set the port for each Docker repository defined in your system, and only then generate the reverse proxy configuration. Note also that each repository must be bound to a unique port

Best Practice

We recommend creating a **Docker Virtual Repository** which aggregates all of your other Docker repositories, and use that to pull and push images. This way you only need to set up the NGINX configuration for that virtual repository.

Docker Reverse Proxy Settings in Reverse Proxy Configuration	Corresponding Reverse Proxy settings in Docker Repository Advanced Configuration
--	--

Docker Reverse Proxy Settings

Reverse Proxy Method

Port

① When using ports method each Docker repository should be bound to a specific port. Configure the port in each Docker repository under the advanced tab.
Example of docker push or pull command:
`docker pull / push reverse-proxy:<REPOSITORY_PORT>/<IMAGE>:<TAG>`

Reverse Proxy

Registry Name

Registry Port

① To view / download the snippet, go to [reverse proxy configuration](#) page.

REST API

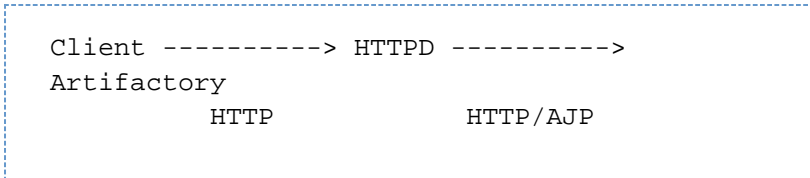
Artifactory also supports managing reverse proxy configuration through the REST API using the following endpoints:

Get Reverse Proxy Configuration	Retrieves the reverse proxy configuration JSON.
Update Reverse Proxy Configuration	.Updates the reverse proxy configuration
Get Reverse Proxy Snippet	Gets the reverse proxy configuration snippet in text format.

Configuring Apache

Setting Up Apache HTTP Server

You can set up Apache HTTP Server as a front end to Artifactory using either the HTTP or AJP protocol.



Using AJP

The AJP protocol offers optimized low-level binary communication between the servlet container and Apache with additional support for smart-routing and load balancing.

The configuration is flexible and can be used either with `mod_proxy_ajp`, or with `mod_jk`.

The example below shows how to configure Apache using `mod_proxy_ajp` which is distributed by default, however you need to install and then enable as follows:

```

Enabling mod_proxy_ajp
sudo a2enmod proxy_ajp
  
```

Configuring Apache With mod_proxy_ajp Installed

The sample virtual host below refers to Apache as a reverse proxy to Tomcat, where Tomcat runs with the AJP connector on port 8019:

Page Contents

- [Setting Up Apache HTTP Server](#)
 - [Using AJP](#)
 - [Configuring Apache With mod_proxy_ajp Installed](#)
 - [Configuring Your Tomcat](#)
 - [Configuring Apache With a Custom Artifactory Path](#)
 - [Using an HTTP Proxy](#)
 - [Configuring Apache With mod_proxy_ajp Installed](#)
- [Setting Up Apache HTTPS](#)
 - [Using AJP](#)
 - [Using an HTTP Proxy](#)
 - [Configuring Apache With mod_proxy_ajp Installed and Tomcat](#)
 - [Configuring a Custom URL Base in Artifactory](#)

Configuring Apache with mod_ajp

```
<VirtualHost *:80>
    ServerAdmin your@email.address.com
    DocumentRoot "/srv/www/httpd/htdocs"
    ServerName artifactory.yourdomain.com
    ErrorLog "logs/artifactory-error_log"
    ProxyPreserveHost on
    ProxyPass /artifactory ajp://<yourdomain>:8019/artifactory
</VirtualHost>
```

Reset Your Cookies

When changing the Artifactory context path in Apache make sure to reset your browser's host and session cookies.

Having a stale context path value cached by cookies can lead to inconsistent issue with the user interface such as `Not authorized to instantiate class` errors when switching between tabs.

Configuring Your Tomcat

If you are using a dedicated Tomcat rather than the one that is bundled with the Artifactory download zip file, you must configure the AJP connector located, by default, under `$CATALINA_HOME/conf/server.xml`:

Configuring a Dedicated Tomcat

```
<Connector port="8019" protocol="AJP/1.3"
    maxThreads="500" minSpareThreads="20"
    enableLookups="false"
    backlog="100" />
```

Please refer to [Apache Tomcat Configuration Reference](#) for more configuration options.

Configuring Apache With a Custom Artifactory Path

You can configure Apache using the same setup as above but here the goal is to have `http://artifactory.yourdomain.com/repository/` as the root URL for Artifactory as follows:

Configuring Apache With Your Custom Artifactory Path

```
<VirtualHost *:80>
  ServerAdmin your@email.address.com
  DocumentRoot "/srv/www/httpd/htdocs"
  ServerName artifactory.yourdomain.com
  ErrorLog "logs/artifactory-error_log"
  ProxyPreserveHost on
  ProxyPass /repository ajp://<yourdomain>:8019/artifactory
  ProxyPassReverse /repository
  http://artifactory.yourdomain.com/artifactory
  ProxyPassReverseCookiePath /artifactory /repository
</VirtualHost>
```

Using an HTTP Proxy

When running Artifactory with Tomcat, we recommend that you set up Apache to proxy Artifactory via HTTP.

You must configure redirects correctly using the PassReverse directive, and also set the base URL in Artifactory itself so that the UI links show up correctly.

Configuring Apache With mod_proxy_ajp Installed

The sample virtual host assumes that the Tomcat HTTP connector runs on port 8081.

Ensuring HTTP Redirect Works Correctly

For HTTP redirects to work, you must set a PassReverse directive on Apache, otherwise the underlying container base URL is passed in redirects

In the example below it is set to `http://artifactory.yourdomain.com/artifactory/`.

Setting a PassReverse Directive on Apache

```
<VirtualHost *:80>
  ServerAdmin your@email.address.com
  DocumentRoot "/srv/www/httpd/htdocs"
  ServerName artifactory.yourdomain.com
  ErrorLog "logs/artifactory-error_log"
  ProxyPreserveHost on
  ProxyPass /artifactory http://<yourdomain>:8081/artifactory
  ProxyPassReverse /artifactory
  http://artifactory.yourdomain.com/artifactory
</VirtualHost>
```

Setting Up Apache HTTPS

You can set up Apache with SSL (HTTPS) as a front end to Artifactory using either the HTTP or AJP protocol.

```
Client -----> HTTPS -----> HTTPD -----> HTTP/AJP -----> Artifactory
```

Using AJP

If you are not running Artifactory with Tomcat, then it is recommended to use AJP since it provides the servlet container with all the information about the correct base URL and requires no configuration in Artifactory.

Using an HTTP Proxy

Configuring Apache With `mod_proxy_ajp` Installed and Tomcat

The Apache and Tomcat sample configuration is as described in the section on Apache HTTP Server above under [Using AJP](#).

Configuring a Custom URL Base in Artifactory

When using an HTTP proxy, the links produced by Artifactory, as well as certain redirects contain the wrong port and use the `http` instead of `https`.

Therefore, you must configure a custom base URL as follows:

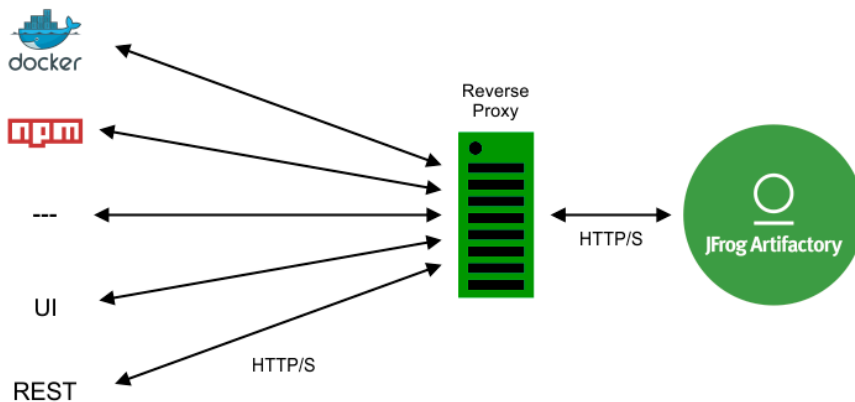
1. On the **Admin** tab select **Configuration | General** *Custom URL Base* field.
2. Set the **Custom URL Base** field to the value used to contact Artifactory on Apache
For example: `https://artifactory.yourdomain.com/artifactory`

Please refer to [General Configuration](#) for more details about configuring the base URL.

Configuring NGINX

Setting Up the NGINX Server

You can use Artifactory behind an nginx server.



When setting up nginx as a front end to Artifactory it is recommended to use HTTP or HTTPS.

Using HTTP or HTTPS

You must set the base URL in Artifactory itself so that the links in the user interface appear correctly.

In the example below, the configuration assumes that the Tomcat HTTP connector runs on port 8081.

Page Contents

- [Setting Up the NGINX Server](#)
 - [Using HTTP or HTTPS](#)
 - [Configuring a Custom URL Base in Artifactory](#)
 - [Advanced Tomcat Configuration](#)

Configuring nginx to use HTTP or HTTPS

```
## add ssl entries when https has been set in config
ssl_certificate      /etc/nginx/ssl/docker.jfrog.com.crt;
ssl_certificate_key  /etc/nginx/ssl/docker.jfrog.com.key;
ssl_session_cache   shared:SSL:1m;
ssl_prefer_server_ciphers   on;
## server configuration
server {
    listen 443 ssl;
    listen 80 ;

    server_name artifactory.jfrog.com;
    if ($http_x_forwarded_proto = '') {
        set $http_x_forwarded_proto $scheme;
    }
    ## Application specific logs
    ## access_log /var/log/nginx/artifactory.jfrog.com-access.log timing;
    ## error_log /var/log/nginx/artifactory.jfrog.com-error.log;
    rewrite ^/$ /artifactory/webapp/ redirect;
    rewrite ^/artifactory/?(/webapp)?$ /artifactory/webapp/ redirect;
    chunked_transfer_encoding on;
    client_max_body_size 0;
    location / {
        proxy_read_timeout 900;
        proxy_pass_header Server;
        proxy_cookie_path ~*^/.*/;
        if ( $request_uri ~ ^/artifactory/(.*)$ ) {
            proxy_pass
http://<rproxy_artifactory>:8081/artifactory/$1;
        }
        proxy_pass http://rproxy_artifactory:8081/artifactory/;
        proxy_set_header X-Artifactory-Override-Base-Url
$http_x_forwarded_proto://$host:$server_port/<public context>;
        proxy_set_header X-Forwarded-Port $server_port;
        proxy_set_header X-Forwarded-Proto $http_x_forwarded_proto;
        proxy_set_header Host $http_host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

Internal Proxies

Regular expression (using `java.util.regex`) that a proxy's IP address must match to be considered an internal proxy. Internal proxies that appear in the `remoteIpHeader` are trusted and do not appear in the `proxiesHeader` value.

If not specified, the default value of `10\.\d{1,3}\.\d{1,3}\.\d{1,3}|192\.168\.\d{1,3}\.\d{1,3}|169\.254\.\d{1,3}\.\d{1,3}|127\.\d{1,3}\.\d{1,3}\.\d{1,3}` is used.

Configuring a Custom URL Base in Artifactory

When using an HTTP proxy, the links produced by Artifactory, as well as certain redirects contain the wrong port and use the `http` instead of `https`.

Therefore, you must configure a custom base URL as follows:

1. On the **Admin** tab select **Configuration | General** *Custom URL Base* field.
2. Set the **Custom URL Base** field to the value used to contact Artifactory on Apache
For example: <https://artifactory.yourdomain.com/artifactory>

Please refer to [General Configuration](#) for more details about configuring the base URL.

Advanced Tomcat Configuration

On Tomcat you may modify your HTTP connector configuration to support advanced capabilities, for example:

Configuring the HTTP connector

```
<Connector port="8081" protocol="HTTP/1.1"
  maxThreads="500" minSpareThreads="20"
  enableLookups="false" disableUploadTimeout="true"
  backlog="100"/>
```

HTTP connector location

By default, the HTTP Connector can be found in `$(CATALINA_HOME)/conf/server.xml`

Mail Server Configuration

Overview

Artifactory supports sending mail to notify administrators and other users for significant events that happen in your system.

Some examples are:

- Watch notifications
- Alerts for backup warnings and errors
- License violation notifications

To enable mail notifications, you need to configure Artifactory with your mail server details as described below.

Page Contents

- [Overview](#)
- [Setup](#)

Setup

To access the mail server configuration, in the **Admin** module select **Configuration | Mail**.

Setup is straightforward and can be verified by sending a test message. Simply click "Send Test Mail" in the **Configure Mail** screen.

artifactory By JFrog Welcome, Admin (Log Out) [Help](#)

- Admin
- Back >
- Configuration
- General
- Repositories
- Repository Layouts
- Artifactory Licenses
- Black Duck
- Property Sets
- Proxies
- Mail**
- Bintray
- Register License
- Security
- Services
- Import & Export

Configure Mail

Mail Server Settings

Enable ?

Host * Port *

Username Password

From Subject Prefix

Artifactory URL

Use TLS

Use SSL

Test Message Recipient

Enabled	When set, mail notifications are enabled
Host	The host name of the mail server
Port	The port of the mail server
Username	The username for authentication with the mail server
Password	The password for authentication with the mail server
From (optional)	The "from" address header to use in all outgoing mails.
Subject Prefix	A prefix to use for the subject of all outgoing mails
Artifactory URL (optional)	The Artifactory URL to use in all outgoing mails to denote links to Artifactory.
Use TLS	When set, uses Transport Layer Security when connecting to the mail server

Use SSL	When set, uses a secure connection to the mail server
Test Message Recipient	The email address of a recipient to receive a test message

Configuration Files

All Artifactory configuration files are located under the `$ARTIFACTORY_HOME/etc` folder.

On Linux, Solaris and MacOS `$ARTIFACTORY_HOME` is usually a soft link to `/etc/artifactory`.

Global Configuration Descriptor

The global Artifactory configuration file is used to provide a default set of configuration parameters.

The file is located in `$ARTIFACTORY_HOME/etc/artifactory.config.xml` and is loaded by Artifactory at initial startup. Once the file is loaded, Artifactory renames it to `artifactory.config.bootstrap.xml` and from that point on, the configuration is stored internally in Artifactory's storage. This ensures Artifactory's configuration and data are coherently stored in one place making it easier to back up and move Artifactory when using direct database backups. On every startup, Artifactory also writes its current configuration to `$ARTIFACTORY_HOME/etc/artifactory.config.startup.xml` as a backup.

At any time, the default configuration can be changed in the Artifactory UI **Admin** module.

There are two ways to directly modify the Global Configuration Descriptor:

1. Using the Artifactory UI
2. Using the REST API

Page Contents

- Global Configuration Descriptor
 - Modifying Configuration Using the UI
 - Modifying Configuration Using the REST API
 - Bootstrapping the Global Configuration
- Security Configuration Descriptor
 - Modifying Security Using the UI
 - Modifying Security Using the REST API
 - Bootstrapping the Security Configuration
- Content Type/MIME Type
 - MIME Type Attributes
 - Setting Content-Type During Download
- System Properties
- Logging Configuration Files
- Storage Properties

Care

Direct modification of the global configuration descriptor is an advanced feature, and if done incorrectly may render Artifactory in an undefined and unusable state. We strongly recommend backing up the configuration before making any direct changes, and taking great care when doing so.

Modifying Configuration Using the UI

You can access the Global Configuration Descriptor in the **Admin** module under **Advanced | Config Descriptor**. There you can modify the file's contents directly or copy the contents from the entry field.

Config Descriptor

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <config xmlns="http://artifactory.jfrog.org/xsd/1.6.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.jfrog.org/xsd/artifactory-v1_6_0.xsd">
3   <serverName>Arti4-Demo</serverName>
4   <offlineMode>false</offlineMode>
5   <fileUploadMaxSizeMb>101</fileUploadMaxSizeMb>
6   <dateFormat>dd-MM-yy HH:mm:ss z</dateFormat>
7   <addons>
8     <showAddonsInfo>true</showAddonsInfo>
9     <showAddonsInfoCookie>1344369836164</showAddonsInfoCookie>
10  </addons>
11  <mailServer>
12    <enabled>true</enabled>
13    <host>smtp.gmail.com</host>
```

Cancel

Save

Modifying Configuration Using the REST API

You can retrieve or set the global configuration by sending a GET or POST request to `http://<host>:<port>/artifactory/api/system/configuration`. For example:

Retrieving and Setting the Global Configuration Descriptor

```
curl -u admin:password -X GET -H "Accept: application/xml"
http://localhost:8080/artifactory/api/system/configuration
curl -u admin:password -X POST -H "Content-type:application/xml"
--data-binary @artifactory.config.xml
http://localhost:8080/artifactory/api/system/configuration
```

Bootstrapping the Global Configuration

You can bootstrap Artifactory with a predefined global configuration by creating an `$ARTIFACTORY_HOME/etc/artifactory.config.import.xml` file containing the Artifactory configuration descriptor.

If Artifactory detects this file at startup, it uses the information in the file to override its global configuration. This is useful if you want to copy the configuration to another instance of Artifactory.

Security Configuration Descriptor

There are two ways to directly modify the Security Configuration Descriptor:

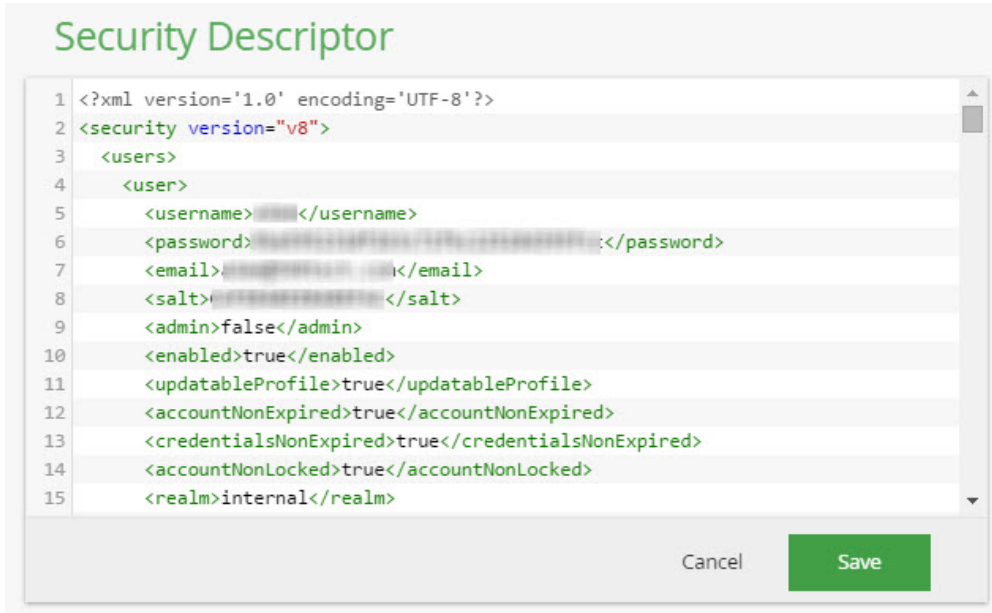
1. Using the Artifactory UI
2. Using the REST API

Care

Direct modification of the security descriptor is an advanced feature, and if done incorrectly may render Artifactory in an undefined and unusable state. We strongly recommend backing up the configuration before making any direct changes, and taking great care when doing so.

Modifying Security Using the UI

You can access the Security Configuration Descriptor in the **Admin** module under **Advanced | Security Descriptor**. There you can modify the file's contents directly or copy the contents from the entry field.



Modifying Security Using the REST API

You can retrieve or set the security configuration by sending a GET or POST request to `http://<host>:<port>/artifactory/api/system/security`. For example:

```
curl -u admin:password -X GET -H "Accept: application/xml"
http://localhost:8080/artifactory/api/system/security
curl -u admin:password -X POST -H "Content-Type: application/xml"
--data-binary @security.xml
http://localhost:8080/artifactory/api/system/security
```

Admin privileges

You must supply a user with **Admin** privileges to modify the security descriptor through the REST API

Bootstrapping the Security Configuration

Artifactory stores all security information as part of its internal storage. You can bootstrap Artifactory with a predefined security configuration by creating a `$ARTIFACTORY_HOME/etc/security.import.xml` file containing the Artifactory exported security configuration information.

If Artifactory detects this file at startup, it uses the information in the file to override all security settings. This is useful if you want to copy the security configuration to another instance of Artifactory.

Content Type/MIME Type

Artifactory provides a flexible mechanism to manage content type/MIME Type. You can define system-wide MIME types for common usage, but you can also overwrite the MIME types for specific files as needed. The list of default MIME types can be found in `$ARTIFACTORY_HOME/etc/mimetypes.xml` and can be edited in order to add, remove or change MIME types. If a file has an extension that is not supported by any of the MIME types, or does not have an extension at all, Artifactory will use the default MIME type of `application/octet-stream`. To determine an artifact's MIME type, Artifactory compares its extension with the those in the `mimetype.xml` file, and applies the MIME type of the first extension that matches.

MIME Type Attributes

Each MIME type may have the following attributes:

type	The MIME type unique name (mandatory)
extensions	A comma separated list of file extensions mapped to this MIME type (mandatory)
index	True if this MIME type should be indexed for archive searching (valid only for supported archive files)
archive	True if this MIME type is a browsable archive
viewable	True if this MIME type can be viewed as a text file inside Artifactory UI
syntax	The UI highlighter syntax to for this MIME type (only relevant if this is a viewable type)
css	The css class of a display icon for this mime type

Example of mimetype.xml

```
<mimetypes version="4">
  <mimetype type="text/plain" extensions="txt, properties, mf, asc"
viewable="true" syntax="plain"/>
  <mimetype type="text/html" extensions="htm, html" viewable="true"
syntax="xml"/>
  <mimetype type="text/css" extensions="css" viewable="true"
syntax="css"/>
  <mimetype type="text/xsl" extensions="xsl" viewable="true"
syntax="xml"/>
  <mimetype type="text/xslt" extensions="xslt" viewable="true"
syntax="xml"/>
  <mimetype type="text/x-java-source" extensions="java" viewable="true"
syntax="java"/>
  <mimetype type="text/x-javafx-source" extensions="fx" viewable="true"
syntax="javafx"/>
</mimetypes>
```

For example, from the extensions parameter in the above *mimetypes.xml* file sample we can conclude that:

- *test.properties* is a *text/plain* MIME type
- *test.css* is a *text/css* MIME type
- *test.doc* is an *application/octet-stream* MIME type since "doc" is not included in any of the other MIME types).

IMPORTANT: Make sure you restart Artifactory for your changes to take affect.

Artifactory MIME Types

Some of the Mime-Types specified in *mimetypes.xml* (e.g. *application/x-checksum*) are used by Artifactory. Great care should be taken before changing these Mime-Types to ensure Artifactory continues to function correctly.

Setting Content-Type During Download

Using Artifactory, when downloading files you can override the *Content-Type* HTTP header by setting the *artifactory.content-type* property.

If the *artifactory.content-type* property is not explicitly set, Artifactory will use the default mechanism of matching the artifact name extension to the extensions in the *mimetypes.xml* file to apply the Content-Type.

This feature is only available with Artifactory Pro.

System Properties

Rather than configuring properties in the JVM runtime configuration of the hosting container, you can edit *\$ARTIFACTORY_HOME/etc/artifactory.system.properties* file and restart Artifactory.

The Artifactory system properties are documented within this file.

Since these settings impact the entire container VM, we recommend using this feature primarily for specifying Artifactory-related properties only (such as changing the database used by Artifactory, etc.).

Setting properties in *artifactory.system.properties* is an advanced feature and is typically not required.

Do not confuse these setting with those in the *\$ARTIFACTORY_HOME/data/artifactory.properties* file, which are for internal use.

Logging Configuration Files

Artifactory uses the [Logback Framework](#) to manage logging and lets you configure the verbosity of log files. For details please refer to [Configuring Log Verbosity](#)

Storage Properties

Artifactory provides you with a *binarystore.xml* file so that you can configure the specific storage solution used in your system. For details please refer to [Configuring the Filestore](#).

Exposing Maven Indexes

Overview

Artifactory exposes Maven indexes for use by Maven integrations of common IDEs (for example, IntelliJ IDEA, NetBeans, Eclipse).

Indexes are fetched remotely from remote repositories that provide them and are calculated for local and virtual repositories (note that many repositories do not provide indexes, or do not keep an updated index).

If Artifactory cannot find a remote index, it calculates one locally, based on the remote repository's previously cached artifacts.

Page Contents

- [Overview](#)
- [Usage](#)

Artifactory's search and indexing facilities are not related to Maven indexes

The indexing performed by Artifactory is secure, immediately effective and supports a larger variety of search options, including custom metadata searches.

Maven indexes only exist in Artifactory for the purpose IDE integrations. They are periodically calculated, contain a limited set of data and are non-secure by design.

Information about the content of a repository is exposed to anyone with access to the repository's index, regardless of any effective path permission you have in place. If this is a concern, do not expose an index for that repository.

Using Artifactory SaaS?

The Maven Indexer service is only available on Artifactory SaaS **dedicated servers**.

Usage

To administer Maven indexes, in the **Admin** module select **Services | Maven Indexer**.

Artifactory provides you with controls to specify how frequently indexing is run and which repositories are included in the index calculation.

Maven Indexer Support

Enabled

Cron Expression*

0 23 5 * * ?

Next Indexing Time

Thu Jul 09 05:23:00 UTC 2015

Run Indexing Now

Included Repositories

filter...

gradle-libs

java.net.m1

repo1

Excluded Repositories

RubyGems-local x

dfsfq x

ext-release-local x

ext-snapshot-local x

ivy-local x

libs-release-local x

Clear All

Cancel Save

Enabled	When set, indexing is enabled and will run according to the Cron Expression setting
Cron Expression	A valid Cron expression that determines the frequency in which Maven indexes on the selected repositories will be recalculated
Next Indexing Time	Indicates the next scheduled indexing run
Run Indexing Now	Invokes indexing immediately

Included Repositories	Specifies the repositories that should be indexed on the next run
Excluded Repositories	Specifies the repositories that should not be indexed on the next run
Clear All	Removes all repositories from the Included Repositories list

Indexing is resource intensive

Calculating and indexing for a repository may be a resource intensive operation, especially for a large local repository or if the repository is a virtual one containing other underlying repositories.

Therefore, we recommend that you do not include repositories that do not require indexing for a periodic index calculation.

Clustering Artifactory

Active/Active Architecture

Artifactory HA is an Active/Active clustered installation of Artifactory that provides a full set of true High Availability features and is supported with an Artifactory [Enterprise License](#).

For full details please refer to [Artifactory High Availability](#).

Active/Passive Architecture

Overview

Artifactory clustered Active/Passive architecture provides fast disaster recovery and can be implemented in one of the following two ways:

- [Deployment on fault-tolerant storage](#) (strongly recommended)
- [Periodic cross-server data sync](#).

Deployment on Fault-tolerant Storage

Using a fault-tolerant disk mounted on another machine allows for a very short MTR (Mean Time to Recovery) in case the "active" server goes down. If Artifactory is deployed on a NAS or SAN the "passive" machine can immediately mount the storage, bootstrap Artifactory from it and start accepting requests in place of the originally "active" machine that is has gone down.

Page Contents

- [Active/Active Architecture](#)
- [Active/Passive Architecture](#)
 - [Overview](#)
 - [Deployment on Fault-tolerant Storage](#)
 - [Cross-server Data Synchronization](#)
 - [Synchronizing the Data and Configuration](#)

- on Directories
- Synchronizing the Database
- Time Synchronization on the Standby Server

To set this up quickly and efficiently, we recommend using the built-in Virtual Machine Failover feature offered by virtualization software providers as follows:

1. Create a VM image that runs the Artifactory startup script and mounts the auxiliary storage.
2. The storage should contain the full Artifactory installation along with the data in a location defined as `$ARTIFACTORY_HOME`.
3. Use the VM image on two Virtual Machines and have Artifactory running on one machine while the other machine is readily available as a failover target by the virtualization monitor.

Cross-server Data Synchronization

If deployment on fault-tolerant storage, as described in the previous section, is not possible (or if redundancy is required), fault-tolerance can be achieved by correctly replicating the data folder to a warm standby server.

The setup of an up-to-date passive replication server for the active Artifactory server requires database replication and synchronization of file system directories.

Synchronizing the Data and Configuration Directories

To synchronize the data and configuration directories you need to run `rsync` on `$ARTIFACTORY_HOME/data` and `$ARTIFACTORY_HOME/etc`.

This can be done by running the `rsync` command on `$ARTIFACTORY_HOME` while excluding the directories that are not required as follows:

```
rsync -vvhah --del --progress --log-file=/home/replication/replication.log
--exclude-from=rsync-excludes.txt \
artifactory@active-artifactory-host:$ARTIFACTORY_HOME/ $ARTIFACTORY_HOME/
```

For the above example the `rsync-excludes.txt` file appears as follows:

```
/work/
/data/tmp*/
/data/cache/
/logs/
```

rsync

The `rsync` should be executed from the passive stand-by server

Synchronizing the Database

Database Replication

Database replication must run **before** executing `rsync`.

The procedure to synchronize a database varies between the different database vendors. Please refer to the relevant documentation for your specific database.

For example, instructions on how to synchronize with MySQL can be found in the MySQL documentation for [How to Set Up Replication](#).

It is also possible to use a full dump/restore procedure on the database to synchronize the database and filestore state. In this case, we recommend that you perform the dump in a single routine along with *rsync* (in case of File System Storage Types).

Time Synchronization on the Standby Server

It is very important that the metadata stored in the database and the data stored on the file system are synchronized on the standby server.

A straightforward way to achieve this, is to make sure that the database synchronized is in a state that is **prior** to the file system (data/filestore) state.

This allows you to:

- Make a database dump before executing the file system sync,
- Activate database replication on demand just before executing *rsync*.

Since the sync operations are not atomic, there may be a gap between the data from *rsync* and data from database replication.

1. The snapshot time that Artifactory is set to is the database replication time.
2. Items synced to the file system which have no representation in the database can be purged by clicking on **Prune Unreferenced Data** in the **Admin** tab and then **Advanced | Maintenance** in the Artifactory configuration.

System Monitoring and Maintenance

Overview

Artifactory provides a set of tools that allow you to monitor and maintain your system to keep it running and responsive:

- [System Information](#) lets you examine the various properties and parameters of your system at runtime and is a valuable resource when investigating any issues that may arise.
- You can [monitor storage](#) to view the number of artifacts and physical files in your system as well as the amount of space that they occupy.
- [Log files](#) let you monitor all the activity that has occurred in your system
- [JMX Beans](#) let you monitor repositories, executor pools and storage
- You can configure regular, periodic [maintenance operations](#) to manage resource allocation and free up disk space
- You can define a regimen for [complete system backup](#)
- You can [import and export](#) data both at system level and repository level
- You can monitor activity related to a specific artifact by [defining a watch](#)

Read More

- [System Information](#)
- [Monitoring Storage](#)
- [Artifactory Log Files](#)
- [Artifactory JMX MBeans](#)
- [Regular Maintenance Operations](#)
- [Managing Backups](#)
- [Importing and Exporting](#)
- [Managing Disk Space Usage](#)
- [Getting Support](#)

System Information

Overview

Artifactory can display different system information such as JVM runtime parameters, JVM arguments, memory usage and more.

This can be useful if you need to examine your system at runtime and is a valuable resource when investigating any issues that may arise.

To view Artifactory system information, in the **Admin** module, go to **Advanced | System Info**.

Page Contents

- [Overview](#)

System Info

Storage Info:

Database Type:	derby
Storage Type:	filesystem
Connection Url:	jdbc:derby:/data/downloads/artifactory-powerpack-4.x-SNAPSHOT/data/derby;create=true

System Properties:

artifactory.running.mode:	PRO
artifactory.running.state:	Online
Wicket_HeaderRenderStrategy:	org.apache.wicket.markup.renderStrategy.ParentFirstHeaderRenderStrategy
artifactory.home:	/data/downloads/artifactory-powerpack-4.x-SNAPSHOT
artifactory.version:	4.x-SNAPSHOT
awt.toolkit:	sun.awt.X11.XToolkit
catalina.base:	/data/downloads/artifactory-powerpack-4.x-SNAPSHOT/tomcat
catalina.home:	/data/downloads/artifactory-powerpack-4.x-SNAPSHOT/tomcat
catalina.useNaming:	true
common.loader:	"\${catalina.base}/lib","\${catalina.base}/lib/*.jar","\${catalina.home}/lib","\${catalina.home}/lib/*.jar"
derby.language.logStatementText:	false
derby.module.mgmt.jmx:	org.apache.derby.impl.services.jmxnone.NoManagementService
derby.storage.pageCacheSize:	500
derby.stream.error.file:	/data/downloads/artifactory-powerpack-4.x-SNAPSHOT/logs/derby.log
derby.stream.error.logSeverityLevel:	0
file.encoding:	UTF8
file.encoding.pkg:	sun.io
file.separator:	/

Monitoring Storage

Overview

Artifactory allows you to monitor various statistics related to the amount of storage that repositories occupy in your system. You can view the number of artifacts and physical files as well as the amount of space that they occupy. To monitor usage of storage in your system, in the **Admin** module, go to **Advanced | Storage Summary**.

Page Contents

- [Overview](#)
- [Binaries](#)
- [File Store](#)
 - [Filestore Sharding](#)
- [Repositories](#)

Binaries

This section provides information on the number of files in your system and the amount of physical and virtual storage that they occupy.

Binaries

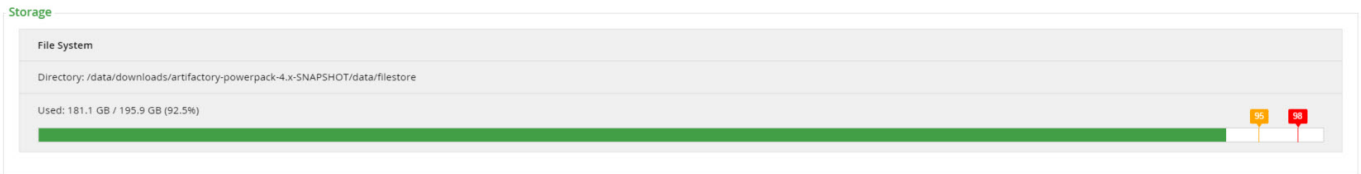
Binaries Size:	6.68 GB	Binaries Count:	40,974
Artifacts Size:	6.71 GB	Artifacts Count: ?	51,191
Optimization: ?	99.53%	Items Count: ?	64,600

Binaries Count	The total number of physical binaries stored in your system.
Binaries Size	The amount of physical storage occupied by the binaries in your system.
Artifacts Size	The amount of physical storage that would be occupied if each artifact was a physical binary (not just a link).
Optimization	The ratio of Binaries Size to Artifacts Size . This reflects how much the usage of storage in your system has been reduced by Artifactory
Items Count	The total number of items (both files and folders) in your system.

File Store

Your system is set up to store binaries as defined in your [storage configuration file](#).

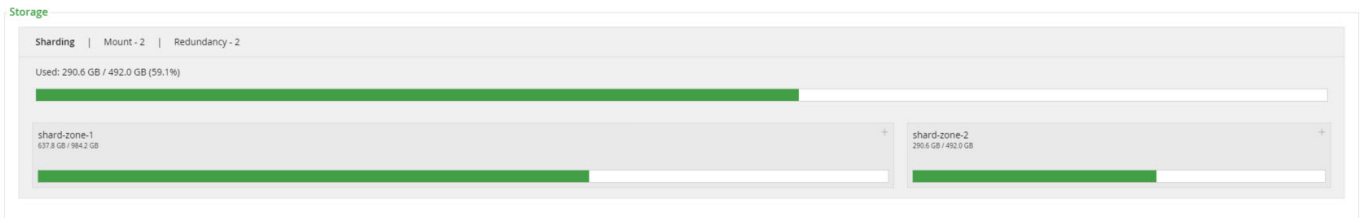
This section provides information on where your binaries are stored and the amount of storage space they are using.



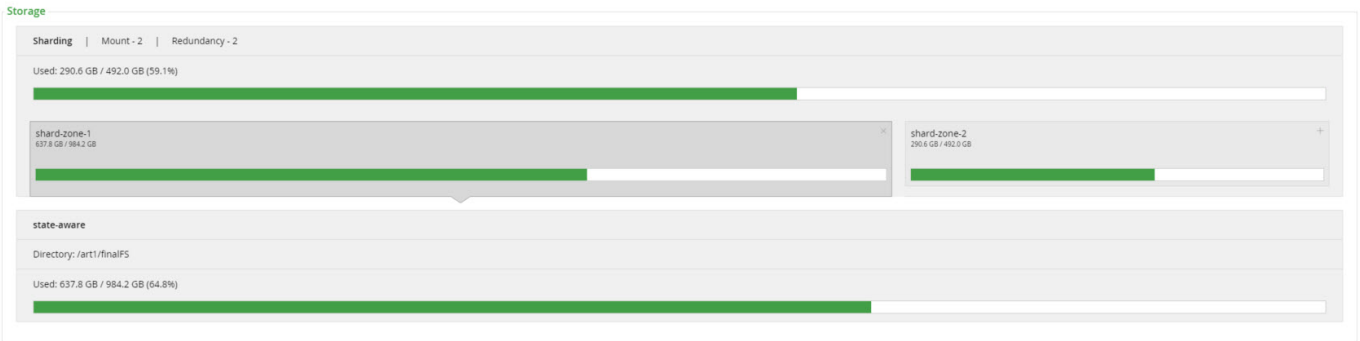
Storage Type	The type of storage used (e.g. "File system").
Storage Directory	If Storage Type is "filesystem" then this is the path to the physical file store. If Storage Type is "fullDb" then this is the path to the directory that caches binaries when they are extracted from the database.
Usage	Displays the amount of storage used out of the total available. Storage space warning and limit thresholds set for your system are also displayed.

Filestore Sharding

From version 4.6, Artifactory offers an additional and highly flexible way to manage storage through [Configuring the Filestore](#). If you use the advanced configuration to setup [filestore sharding](#) in your system, your usage of storage is displayed with details of how sharding is configured.



You can select any of the sharding zones to drill down and get more details about it.



Repositories

The **Repositories** section provides detailed information about the storage used by each repository in your system.

117 Repositories ?

Filter by Repository Key

Page 1 of 5

Repository Key	Repository Type	Package Type	Percentage	Artifacts Size	Files	Folders	Items
TOTAL	N/A	N/A	100%	6.71 GB	51181	13409	64590
Trash Can	N/A	🗑️ Trash	0%	0 bytes	0	0	0
testConn-local	LOCAL	m Maven	47.18%	3.16 GB	80	0	80
bower-local	LOCAL	📦 Bower	39.1%	2.62 GB	49678	12687	62365
rpm-local	LOCAL	📦 YUM	2.82%	193.88 MB	22	14	36
npm-virtual	VIRTUAL	📦 Npm	2.63%	180.49 MB	1	1	2
npm-local	LOCAL	📦 Npm	1.82%	124.79 MB	4	4	8
plugins-release-local	LOCAL	m Maven	1.39%	95.80 MB	346	28	374
p2-local	LOCAL	m Maven	0.94%	64.76 MB	61	24	85
docker-dev-local2	LOCAL	📦 Docker	0.92%	63.45 MB	8	4	12
dockerv1	LOCAL	📦 Docker	0.91%	62.80 MB	19	13	32

Repository Key	The repository id.
Repository Type	Indicates if this is a local repository, remote repository cache or a virtual repository.
Package Type	The repository's package type.
Percentage	The percentage of the total available space occupied by this repository.
Artifacts Size	The amount of space used by artifacts in this repository. Similar to the total export size (including non-unique artifact references).

Files	The total number of files in this repository.
Folders	The total number of folders in this repository.
Items	The total number of items (folders and files) in this repository.

Artifactory Log Files

Overview

Artifactory uses the [Logback Framework](#) to manage logging. Activity is logged according to type in four different log files which can be found under the `ARTIFACTORY_HOME/logs` folder.

The following log files are available:

artifactory.log	The main Artifactory log file containing data on Artifactory server activity.
access.log	Security log containing important information about accepted and denied requests, configuration changes and password reset requests. The originating IP address for each event is also recorded.
request.log	Generic http traffic information similar to the Apache HTTPd request log.
import.export.log	A log used for tracking the process of long-running import and export commands.
sha256_migration.log	Logs status and errors when migrating the Artifactory database to include SHA256 values .

Page Contents

- [Overview](#)
- [Configuring Log Verbosity](#)
 - [Minimizing Output to catalina.out](#)
- [Log File Structure](#)
 - [Request Log](#)
 - [Access Log](#)
- [Viewing Log Files from the UI](#)
- [Sending Artifactory Logs to Syslog](#)

Tomcat/Servlet container-specific log files

When running Artifactory inside an existing servlet container, the container typically has its own log files.

These files normally contain additional information to that in `artifactory.log` or application bootstrapping-time information that is not found in the Artifactory logs.

In Tomcat, these files are `catalina.out` and `localhost.yyyy-mm-dd.log` respectively.

Configuring Log Verbosity

The verbosity of any logger in your system can be configured by entering or modifying the `level` value in the corresponding entry in the Logback configuration file `ARTIFACTORY_HOME/etc/logback.xml`.
For example:

Modifying the verbosity of a logger

```
<logger name="org.apache.wicket">
  <level value="error"/>
</logger>
```

Artifactory loads any changes made to the Logback configuration file within several seconds without requiring a restart.

Minimizing Output to catalina.out

When running Artifactory as a background service, Artifactory log messages are redirected to `catalina.out` which may cause this file to be over-inflated with content. To reduce the volume of logging to `catalina.out` we recommend adding a "threshold filter" to the "CONSOLE" appender in `logback.xml` as follows:

```
...

<appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">

  <!-- Add a Threshold filter to reduce log output that is below the
  specified threshold. In the example below, only ERROR level log messages
  will be added -->
  <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
    <level>ERROR</level>
  </filter>

  <encoder class="ch.qos.logback.core.encoder.LayoutWrappingEncoder">
    <layout
class="org.artifactory.logging.layout.BackTracePatternLayout">
      <pattern>%date ${artifactory.contextId}[%thread] [%-5p]
\(%-20c{3}:%L\) - %m%n</pattern>
    </layout>
  </encoder>
</appender>

...
```

Log File Structure

The Request and Access log files each display specific type of activity and as such have a consistent and specific file structure for maximum readability

Request Log

A request log file record has the following structure:

Date and Time stamp | Request time | Request type | IP | User name | Request method | Requested resource path | Protocol version | Response code | Request Content-Length

Note: If not provided by the client, 'Request Content-Length' is initialised as "-1".

Here is a typical example:

```
Request log file record sample  
20140508154145 | 2632 | REQUEST | 86:12:14:192 | admin | GET | /jcenter/org/iostreams/  
iostreams/0.2/iostreams-0.2.jar | HTTP/1.1 | 200 | 8296
```

Date and time stamp	The date and time the request was completed and entered into the log file. Format is [YYYYMMDDHHMMSS]
Request time	The time in ms taken for the request to be processed
Request type	DOWNLOAD for a download request UPLOAD for an upload request REQUEST for any other request
IP	The requesting user's IP address
User name	The requesting user's user name or "non_authenticated_user" when accessed anonymously
Request method	The HTTP request method. e.g. GET, PUT etc.
Requested resource path	Relative path to the requested resource
Protocol version	The HTTP protocol version
Response code	The HTTP response code
Size (bytes) of request or response	If request method is GET: Size of response If request method is PUT or POST: Size of request

Access Log

An access log file record has the following structure:

Date and Time stamp | Action response and type | Repository path (Optional) | Message (Optional) | User name | IP

Here is a typical example:

```
Access log file record  
2014-05-08 15:52:27,456 [ACCEPTED DOWNLOAD]  
jcenter-cache:org/iostreams/iostreams/0.2/iostreams-0.2.jar for  
anonymous/86:12:14:192.
```

Date and Time stamp	The date and time that the entry was logged. Format is [YYYY-MM-DD HH:MM:SS, milliseconds]
[Action response and type]	The response (ACCEPTED/DENIED) and the action type (e.g. DOWNLOAD, UPLOAD etc.)
Repository path (Optional)	The repository that was accessed
Message (Optional)	An optional system message
User name	The accessing user's user name or "anonymous" when accessed anonymously
IP	The accessing user's IP address

Viewing Log Files from the UI

You can view or download any of the Artifactory log files from the UI.

In the **Admin** module, under **Advanced | System Logs**, select the file you want to view from the drop-list. The log tail view is automatically refreshed every few seconds, however can be paused and resumed if you wish to browse the log.

The screenshot shows the 'System Logs' interface in Artifactory. At the top, there's a dropdown menu showing 'artifactory.log' and a button that says 'Refreshing logs in 3 seconds (Pause)'. Below that, it indicates the file was last modified on 'Sun Oct 09 13:04:06 IDT 2016' and provides a 'Download (6.6 MB)' link. The main part of the interface is a scrollable log viewer showing various log entries. The entries start with configuration files being loaded (e.g., 'staging.groovy --> Loaded') and then move to detailed initialization logs for various services like 'InternalFolderPruningService', 'InternalTrafficService', 'StorageAggregationInterceptors', etc. The log ends with 'Artifactory application context is ready.' and 'Artifactory successfully started (23.173 seconds)'.

To save system resources, do not leave the log view open in your browser unnecessarily.

Sending Artifactory Logs to Syslog

Some sites want to consolidate logs into the syslog facility. Switching artifactory to use syslog in addition to, or instead of the standard log files takes a quick edit of a couple of files. Artifactory currently uses the logback library for logging, so that's what needs to be configured.

First edit the `$ARTIFACTORY_HOME/etc/logback.xml` file to send logs to the syslog facility. You need to add an appender to syslog:

```
<appender name="SYSLOG" class="ch.qos.logback.classic.net.SyslogAppender">
<syslogHost>localhost</syslogHost>
<facility>SYSLOG</facility>
<suffixPattern>[%thread] %logger %msg</suffixPattern>
</appender>
```

then you need to add this appender to the output, in the section:

```
<root>
<level value="info"/>
<appender-ref ref="CONSOLE"/>
<appender-ref ref="FILE"/>
</root>
```

add:

```
<appender-ref ref="SYSLOG"/>
```

before the `</root>` line.

Save the file, you will not need to restart artifactory for this to take effect.

Since logback is using internet sockets, you have to make sure your syslog facility accepts them. Modern linux distributions are using the `rsyslog` daemon for syslogging. Ensure that the configuration for internet domain sockets is enabled, either by editing `/etc/rsyslog.conf` and uncommenting:

```
# Provides UDP syslog reception
$ModLoad imudp
$UDPServerRun 514

# Provides TCP syslog reception
$ModLoad imtcp
$InputTCPServerRun 514
```

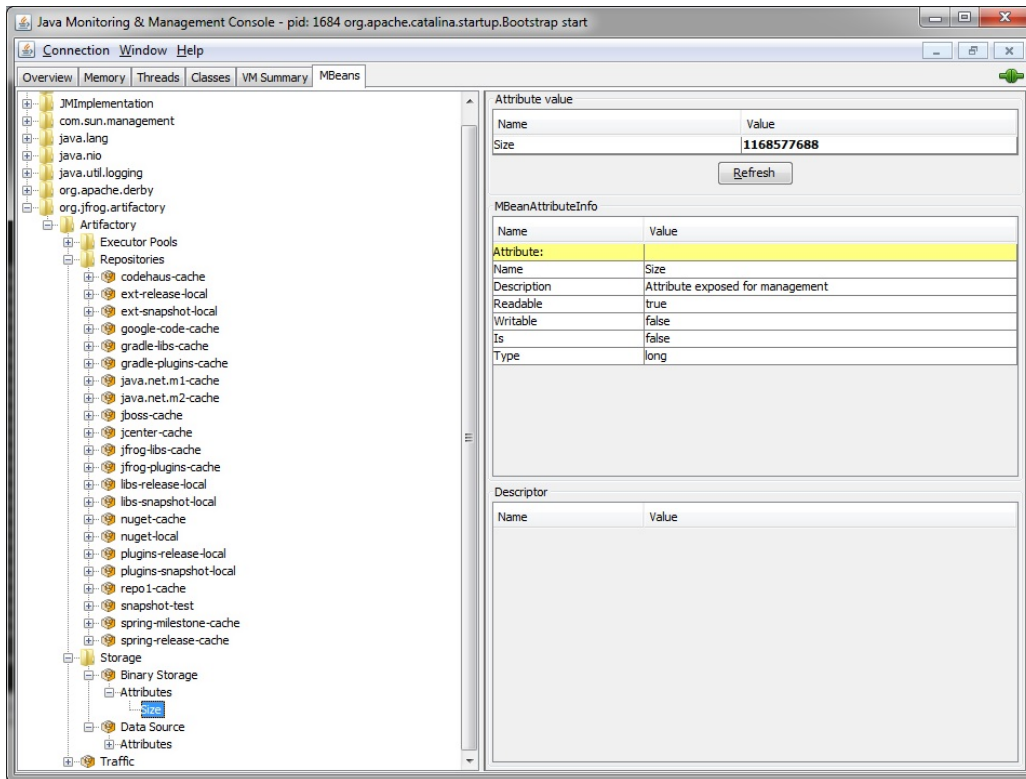
or placing it in a file under `/etc/rsyslog.d` ending in `.conf`

Rsyslog will need restarting with `service rsyslog restart` for this to take effect.

Artifactory JMX MBeans

Overview

Artifactory exposes MBeans under the `org.jfrog.artifactory` domain that let you monitor repositories, executor pools, storage and HTTP connection pools.



Page Contents

- [Overview](#)
- [Repositories](#)
- [Executor Pools](#)
- [Storage](#)
- [HTTP Connection Pools](#)
- [Logging](#)

Repositories

This section lists the available repositories under the current instance of Artifactory. Read-only attributes are as follows:

RepositoryKey	Name of the repository
ArtifactsCount	Number of artifacts in the repository
ArtifactsTotalSize	Total size of all of the artifacts in the repository

Executor Pools

This section lists the executor pools in use by Artifactory. Read-only attributes are as follows:

TaskCount	Total number of tasks that have ever been scheduled for execution
------------------	---

CompletedTaskCount	Total number of tasks that have been completed by the executors
CorePoolSize	Executor pool size
MaximumPoolSize	Maximum size of the executor pool
ActiveCount	Number of active executors
LargestPoolSize	The largest pool size that has been active at any one time

Storage

This section describes File System Binary Storage and Database Data Source read-only attributes.

There is only one File System Binary Storage read-only attribute:

Size	Total size of Artifactory storage in bytes
-------------	--

Database Data Source read-only attributes are:

MaxActive	Maximum number of active connections to the database
Url	Database URL
Idle	Number of idle database connections
ActiveConnectionsCount	Number of active database connections
MaxIdle	Maximum number of idle database connections allowed
MaxWait	Timeout in ms when attempting to get a free connection
MinIdle	Minimum number of idle database connections to maintain

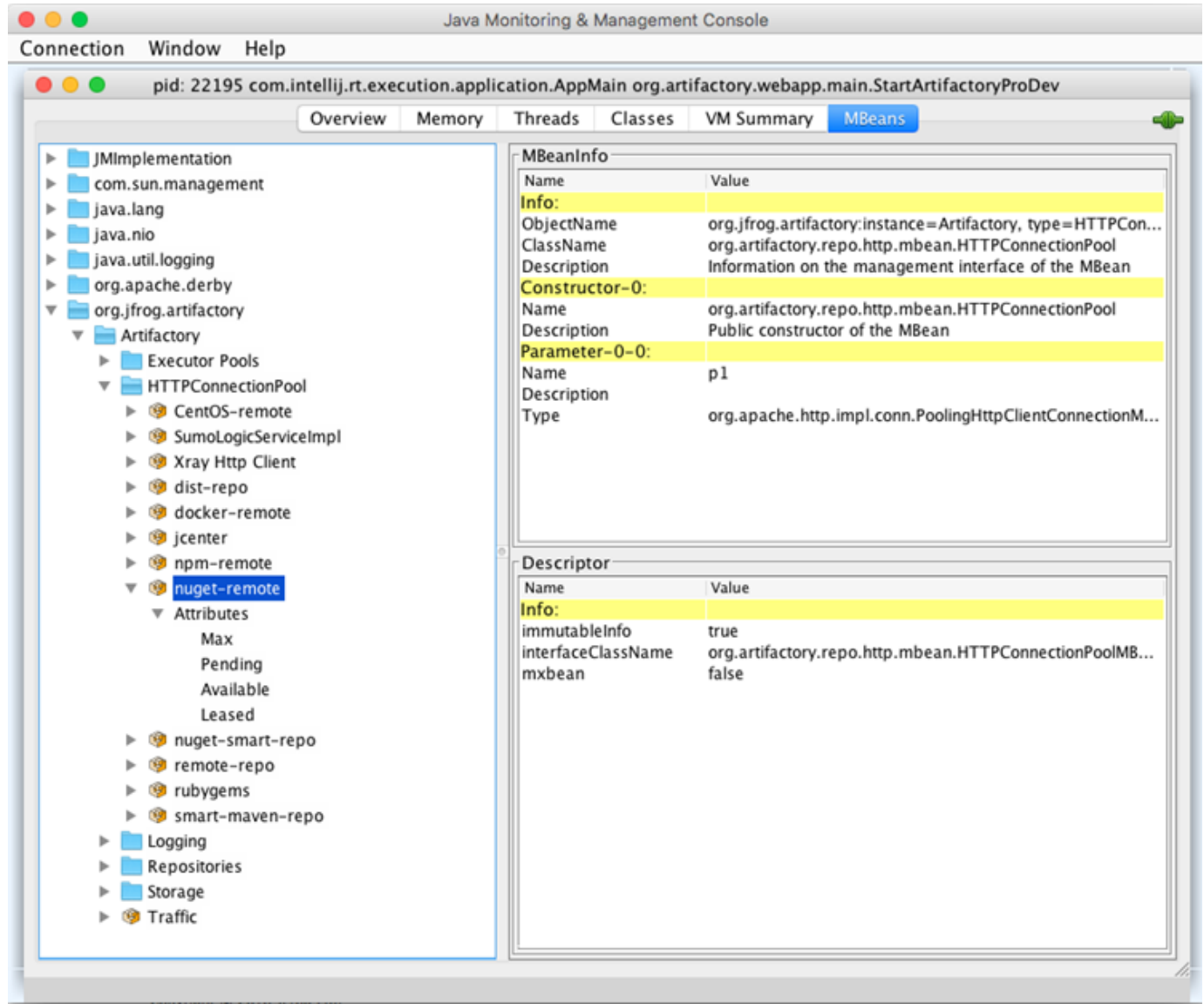
HTTP Connection Pools

Artifactory supports JMX MBeans for the following HTTP resources:

- Remote repositories
- Distribution Repositories
- Xray Client Connection
- Replication Queues
- Event propagation service for Artifactory HA cluster nodes

The following read-only attributes are available for each HTTP connection pool:

Available	Number of available connections
Leased	Number of currently active connections
Max	The maximum number of connections possible
Pending	The number of connections in process and pending completion



Logging

To support log analytics, Artifactory implements log appenders that send log information to Sumo Logic. The following log appenders can be monitored through JMX MBeans:

- Access
- Console

- Request
- Traffic

For each log appender, Artifactory displays the following read-only attributes:

QueueMaxSize	The maximum number of log entries the queue can hold
QueueCurrentSize	The current number of log entries in the queue
QueueDiscardingThreshold	Below this remaining capacity threshold, trace and debug log entries will be discarded
BatchMaxSize	The maximum number of log entries a batch can contain before it is processed
BatchCurrentSize	The size of the current batch pending to be sent to Sumologic
BatchQuietPeriod	The time window (in milliseconds) log entries are collected in a batch before it is being sent to Sumologic
CategoryHeader	The category header value which is sent to Sumologic for this appender

Regular Maintenance Operations

Overview

Artifactory provides several facilities allowing you to maintain your system for optimal performance.

To configure your global system maintenance, in the **Admin** module select **Advanced | Maintenance**.

Artifactory SaaS Users




JFrog manages regular maintenance operations for all instances of Artifactory SaaS. If you are an Artifactory SaaS user, the features described on this page are all monitored and optimally managed for you by JFrog Artifactory SaaS administrators.

Page Contents

- [Overview](#)
- [Garbage Collection](#)
- [Storage Quota Limits](#)
- [Cleanup Unused Cached Artifacts](#)
- [Cleanup Virtual Repositories](#)
- [Storage](#)

Garbage Collection

Garbage Collection

Cron Expression * 	Next Run Time
<input type="text" value="0 0 /4 * * ?"/> 	<input type="text" value="Mon Jul 31 20:00:00 UTC 2017"/>
	

Artifactory uses checksum-based storage to ensure that each binary file is only stored once.

When a new file is deployed, Artifactory checks if a binary with the same checksum already exists and if so, links the repository path to this binary. Upon deletion of a repository path, Artifactory does not delete the binary since it may be used by other paths. However, once all paths pointing to a binary are deleted, the file is actually no longer being used. To make sure your system does not become clogged with unused binaries, Artifactory periodically runs a "Garbage Collection" to identify unused ("deleted") binaries and dispose of them from the datastore. By default, this is set to run every 4 hours and is controlled by a cron expression.

For example, to run garbage collection every 12 hours you should specify the following expression:

```
0 0 /12 * * ?
```

Cron Expression	Specifies the frequency in which garbage collection should be run automatically
Next Run Time	Indicates the next automatic run of garbage collection according to the specified Cron Expression
Run Now	Manually invokes garbage collection immediately

Garbage collection frequency
Garbage collection is a resource intensive operation. Running it too frequently may compromise system performance.

Storage Quota Limits

Storage Quota

<input checked="" type="checkbox"/> Enable Quota Control	
Storage Space Limit (Percentage)*	Storage Space Warning (Percentage)*
<input type="text" value="95"/>	<input type="text" value="85"/>

Artifactory lets you set a limit on how much of your entire system disk space storage may be used to ensure that your server file system capacity is never used up. This helps to keep your system reliable and available.

Once disk space used for storage reaches the specified limit, any attempt to deploy a binary is rejected by Artifactory with a status code of **413 Request Entity Too Large** and a "Datastore disk space is too high" error is displayed at the bottom of the **Maintenance** screen.

When using filesystem storage, the partition checked is the one containing the `$ARTIFACTORY_HOME/data/filestore` directory. When using database blob storage, the partition checked is the one containing the `$ARTIFACTORY_HOME/data/cache` directory.


To help you avoid reaching your disk space quota, Artifactory also allows you to specify a warning level. Once the specified percentage of disk

space is used, Artifactory will log a warning in the `$ARTIFACTORY_HOME/logs/artifactory.log` file and display a "Datastore disk space is too high" warning at the bottom of the **Maintenance** screen.

Enable Quota Control	When set, Artifactory will monitor disk space usage and issue warnings and errors according to the quotas specified in Storage Space Limit and Storage Space Warning
Storage Space Limit	The percentage of available disk space that may be used for storage before Artifactory rejects deployments and issues errors
Storage Space Warning	The percentage of available disk space that may be used for storage before Artifactory issues warnings


Cleanup Unused Cached Artifacts

Cleanup Unused Cached Artifacts

Cron Expression * 

Next Run Time

Tue Aug 01 05:12:00 UTC 2017

 Cleanup Unused Cached Artifacts


When configuring a remote repository, the [Keep Unused Artifacts](#) setting lets you specify how long a cached unused artifact from that repository should be kept before it is a candidate for cleanup. This setting does not immediately clean up the unused cached artifact, but merely marks it for clean up after the specified number of hours. The **Cleanup Unused Cached Artifacts** setting specifies when the cleanup operation should run, and only then unused, cached artifacts marked for cleanup are actually removed from the system.

The cleanup frequency is specified with a [cron](#) expression. For example, to run cleanup every 12 hours you should specify the following expression:

```
0 0 /12 * * ?
```


Cleanup Virtual Repositories

Cleanup Virtual Repositories

Cron Expression * 

Next Run Time

Tue Aug 01 00:12:00 UTC 2017

 Clean Virtual Repositories Now

Virtual repositories use an internal cache to store aggregated metadata such as POM files. The Cleanup Virtual Repositories operation deletes cached POM files that are older than 168 hours (one week)

The cleanup frequency is specified with a [cron](#) expression. For example, to run cleanup every 12 hours you should specify the following expression:

Storage

Storage

Compress the Internal Database

Prune Unreferenced Data

<p>Compress the Internal Database</p>	<p>Derby database only This feature is only relevant when using the internal Derby database</p> <p>A Derby database may typically contain unused allocated space when a large amount of data is deleted from a table or its indices are updated. By default, Derby does not return unused space to the operating system. For example, once a page has been allocated to a table or index, it is not automatically returned to the operating system until the table or index is destroyed.</p> <p>When you invoke this action, Artifactory reclaims unused and allocated space in a table and its indexes thereby compressing the internal database.</p> <p>We recommend running this when Artifactory activity is low, since compression may not be able to complete when storage is busy (in which case the storage will not be affected).</p>
<p>Prune Unreferenced Data</p>	<p>Unreferenced binary files may occur due to running with wrong file system permissions on storage folders, or running out of storage space.</p> <p>When you invoke this action, Artifactory removes unreferenced binary files and empty folders present in the filestore or cache folders.</p> <p>Ensure complete shutdown To avoid such errors, we recommend that you always allow Artifactory to shut down completely</p>

Managing Backups

Complete System Backup

You can automatically and periodically backup the entire Artifactory system. The backup process creates a time-stamped directory in the target backup directory.

To define multiple backups, in the **Admin** module, select **Services | Backups**. Each backup may have its own schedule and repositories to either process or exclude.

Page Contents

- [Complete System Backup](#)

- Restoring a Backup

Backup content is stored in standard file system format and can be loaded into any repository, so that Artifactory never locks you out.

Backing up very large filestores

If you are backing up more than 1TB of storage, please refer to [this article](#) in our [Knowledge Base](#) for instructions.

In the Backups page you may select an existing **Backup** to edit, or click "New" to create a new **Backup**.

Edit backup-daily Backup

Backup Settings

Enabled

Backup Key *

backup-daily

Cron Expression * [?](#)

0 0 2 ? * MON-FRI

Next Backup Time

Wed Oct 19 02:00:00 UTC 2016

Server Path For Backup [?](#)

Browse

Advanced

Send Mail to Admins if there are Backup Errors [?](#)

Exclude Builds

Exclude New Repositories

Verify enough disk space is available for backup [?](#)

Incremental

Retention Period Hours [?](#)

0

Filter...

Excluded Repositories

repo1

debian-flat

debian-local

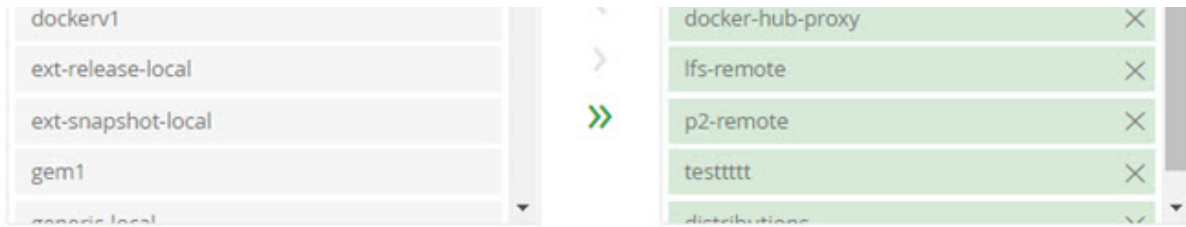
Filter...

Included Repositories

dima

docker-dev-local2

docker-prod-local2



Back up to a Zip Archive (Slow and CPU Intensive)

Backup Key	A unique logical name for this backup
Cron Expression	A valid Cron expression that you can use to control backup frequency. For example, to back up every 12 hours use a value of: <code>0 0 /12 * * ?</code>
Next Time Backup	When the next backup is due to run
Server Path for Backup	The directory to which local repository data should be backed up as files The default is <code>\$ARTIFACTORY_HOME/backup/[backup_key]</code> Each run of this backup will create a new directory under this one with the time stamp as its name.
Send Mail to Admins if there are Backup Errors	If set, all Artifactory administrators will be notified by email if any problem is encountered during backup.
Exclude Builds	Exclude all builds from the backup.
Retention Period	The number of hours to keep a backup before Artifactory will clean it up to free up disk space. Applicable only to non-incremental backups. <div style="border: 1px solid red; padding: 5px; margin-top: 10px;">Do not store any custom files under the target backup directory, since the automatic backup cleanup processes may delete them!</div>
Verify enough disk space is available for backup	If set, Artifactory will verify that the backup target location has enough disk space available to hold the backed up data. If there is not enough space available, Artifactory will abort the backup and write a message in the log file.
Incremental	When set, this backup should be incremental. In this case, only changes from the previous run will be backed up, so the process is very fast. The backup directory name will be called <code>current</code> (as opposed to using the timestamp) The backup files can be used by any incremental file-system based backup utility (such as rsync).
Backup to a Zip Archie (Slow and CPU Intensive)	If set, backups will be created within a Zip archive

Monitoring Backup Progress

During a system backup, Artifactory writes several messages to the `ARTIFACTORY_HOME/logs/artifactory.log` file. To monitor progress of the backup process, look for messages that indicate the beginning and the end of a full system export as in the following example:

```
2016-06-09 02:00:00,023 [art-exec-1] [INFO ]
(o.a.s.ArtifactoryApplicationContext:508) - Beginning full system
export...
...
2016-06-09 02:00:00,357 [art-exec-1] [INFO ]
(o.a.s.ArtifactoryApplicationContext:620) - Full system export
completed successfully.
```

Restoring a Backup

To restore a system backup you need perform a system import. For details please refer to [System Import and Export](#).

Importing and Exporting

Overview

Artifactory supports import and export of data at two levels:

- [System level](#)
- [Repository level](#)

At **system level**, Artifactory can export and import the whole Artifactory server: configuration, security information, stored data and metadata. The format used is identical to the [System Backup](#) format. This is useful when manually running backups and for migrating and restoring a complete Artifactory instance (as an alternative to using database level backup and restore).

At **repository level**, Artifactory can export and import data and metadata stored in a repository. This is useful when moving store data, including its metadata between repositories and for batch population of a repository.

Page Contents

- [Overview](#)
- [System Import and Export](#)
 - [System Import and Export for an HA Cluster](#)
- [Repositories Import and Export](#)
 - [Export](#)
 - [Import](#)
 - [Import Layout](#)

System Import and Export

To access import and export of your entire system, in the **Admin** module, select **Import & Export | System**

System Import & Export

Export System

Export Path on Server *


c:\work

Browse

- Exclude Content
- Exclude Metadata
- Exclude Builds
- Create .m2 Compatible Export [?](#)
- Create a Zip Archive (Slow and CPU Intensive!)
- Output Verbose Log [?](#)

Export

Import System

 This action will wipe all Artifactory content - make sure to back up before completing this action!

Import Zip or Path on Server *

c:\work\20170621.103948

Browse

- Exclude Content
- Exclude Metadata
- Output Verbose Log [?](#)

Import

Target Export Dir	The target directory for the exported files. You may browse your file system to select the directory
Exclude Content	Export: When set, repository binaries are excluded from the export. Import: When set, binaries and metadata are excluded from the import. Only builds and configuration files are imported.

Exclude Metadata	<p>When set, repository metadata are excluded from the import/export. (Maven 2 metadata is unaffected by this setting)</p> <div style="border: 1px solid yellow; padding: 5px; margin-top: 10px;"> <p>Docker repositories must have metadata For Docker repositories to work they must have their metadata intact. Therefore, if you have Docker repositories, make sure that Exclude Metadata is not checked when doing a system export or import.</p> </div>
Exclude Builds	<p>When set, all builds are excluded from the export</p>
Create .m2 Compatible Export	<p>When set, includes Maven 2 repository metadata and checksum files as part of the export</p>
Create a Zip Archive (Slow and CPU Intensive!)	<p>When set, creates and exports to a Zip archive</p>
Output Verbose Log	<p>When set, lowers the log level to "debug" and redirects the output from the standard log to the import-export log.</p> <div style="border: 1px solid blue; padding: 5px; margin-top: 10px;"> <p>Monitoring the log You can monitor the log in the System Logs page.</p> </div>

The source/target of the import/export operations are folders (Zip archives are not recommended) on the Artifactory server itself.

You can use the built-in server-side browsing inside Artifactory to select server-side source/target folders:

Server File System Browser ✕

Mount Point: Path to export

c:\

c:\work

📁

▲
▼

Directory To Export ?

work

Cancel
Select

Importing or exporting a large amount of data may be time consuming. During the import/export operation you can browse away from the page and sample the [System Logs](#) to monitor progress.

System Import and Export for an HA Cluster

When performing a system export and subsequent import for an HA cluster, you need to follow the procedure below to ensure that the cluster is able to correctly synchronize its nodes.

- Perform a normal system export from the source cluster as described [above](#)
- In the target cluster, keep the primary node running, and perform a graceful shutdown of all secondary nodes
- Perform normal system import to the target cluster (which now has only the primary node running) as described [above](#)
- Perform a graceful shutdown of the primary node and then restart it
- [Create the bootstrap bundle](#) on the primary node
- For each secondary node:
 - Delete the following folders
 - `$ARTIFACTORY_HOME/access`
 - `$ARTIFACTORY_HOME/etc/security`
 - `$ARTIFACTORY_HOME/etc/ui`
 - `$ARTIFACTORY_HOME/etc/plugins`
 - Delete the `$ARTIFACTORY_HOME/etc/db.properties` file
 - Delete the `$ARTIFACTORY_HOME/etc/binarystore.xml` file
 - Copy the bootstrap bundle you created on the primary node, `bootstrap.bundle.tar.gz`, to the `$ARTIFACTORY_HOME/etc` folder on the secondary node.

Bootstrap Bundle and db.properties

This is a critical step in the import process. The bootstrap bundle must be installed in each secondary node before you start it up for it to operate correctly in the cluster.

Note also, if the `$ARTIFACTORY_HOME/etc` folder in your secondary node already contains a `db.properties` file, it will prevent the bootstrap bundle from being properly extracted when you start up the secondary node causing the import to fail.

- Start up the secondary node

Once you have completed the import, we recommend verifying that your HA cluster is up and running normally as described in [Testing your HA Configuration](#).

Repositories Import and Export

To access import and export of repositories, in the **Admin** tab, select **Import & Export | Repositories**

Export

Export Repository to Path

Target Local Repository*

All Repositories

Export to Path*

Exclude Metadata

Create .m2 Compatible Export

Output Verbose Log

Export

When exporting, you need to specify the following parameters:

Source Local Repository	You can specify a single repository to export, or All Repositories
Export to Path	The export target directory on your server
Exclude Metadata	When set, repository metadata are excluded from the export.(Maven 2 metadata is unaffected by this setting)
Create .m2 Compatible Export	When set, includes Maven 2 repository metadata and checksum files as part of the export
Output Verbose Log	When set, lowers the log level to "debug" and redirects the output from the standard log to the import-export log. <div style="border: 1px solid #ccc; padding: 5px;"><p>Monitoring the log You can monitor the log in the System Logs page.</p></div>

Import

You can import repositories from a server side folder, or by zipping a repository and uploading it to Artifactory.

Import Repository from Path

Target Local Repository*

debian-local

Server Path For Import*

Browse

Exclude Metadata

Output Verbose Log

Import

When importing, you need to specify the following parameters:

Target Local Repository	You can specify a single repository to import, or All Repositories . The repository layout should be different depending on your selection. Please refer to Import Layout
Server Path for Import	The import source directory on your server
Exclude Metadata	When set, repository metadata are excluded from the import
Output Verbose Log	When set, lowers the log level to "debug" and redirects the output from the standard log to the import-export log. <div style="border: 1px solid #ccc; padding: 5px;"><p>Monitoring the log You can monitor the log in the System Logs page.</p></div>

Don't exclude metadata for Docker

To work with a Docker repository, it must have its metadata intact. Therefore, when importing to/exporting from a Docker repository make sure that **Exclude Metadata** is not checked.

Importing into a Remote Repository Cache

You can take advantage of remote repositories you have already downloaded to your local environment, and import them directly into a local repository.

For example, you can take your local Maven repository (usually located under `~/.m2`) and upload it into Artifactory so that all the artifacts you have already downloaded are now available on the server.

Import Layout

An imported repository needs to be formatted using a Maven 2 repository layout.

When importing a single repository, the file structure within the import folder (or zip file) should be as follows:

```
IMPORT_FOLDER/ZIP_FILE
|
|--LIB_DIR_1
```

When importing all repositories, the file structure within the import folder should be as follows:

```
IMPORT_FOLDER/ZIP_FILE
|
|--REPOSITORY_NAME_DIR_1
| |
| |--LIB_DIR_1
```

When importing all repositories, you need to ensure that the names of the directories representing the repositories in the archive match the names of the target repositories in Artifactory.

Managing Disk Space Usage

Overview

Artifactory includes features to help you manage the amount of disk space used by your system. This is done by providing alerts, limiting the amount of space allocated for the output of automatic procedures, and by cleaning up unused artifacts in a controlled manner.

Garbage Collection

When an Artifactory user "deletes" a file, what is actually deleted is the reference from the Artifactory database to the physical file. Before actually deleting a file Artifactory must scan the system to ensure that there are no other users referencing the file. Scanning the system is very CPU intensive, and locks files while the scan is in process, and this may stress the development environment. Therefore this can be scheduled to run periodically as a "Garbage Collection" process during times when demands on the system are low.

This is done in the Artifactory UI **Admin** module under **Advanced | Regular Maintenance Operations**, where you can schedule an automatic run of Garbage Collection with a **Cron** expression. You can also invoke an immediate run by clicking "Run Storage Garbage Collection".

Garbage Collection

Cron Expression*	Next Garbage Collection Time
<input type="text" value="0 0 /4 * * ?"/>	<input type="text" value="Wed Jul 08 16:00:00 UTC 2015"/>
<input type="button" value="Run Now"/>	

Page Contents

- Overview
 - Garbage Collection
 - Storage Quota
- Limiting the Number of Snapshots
- Deleting Unused Cached Artifacts
- Deleting Complete Versions
- User Plugins

- [Manual Cleanup with the REST API](#)
- [Discarding old builds with Jenkins Artifactory plugin](#)

Storage Quota

To avoid running out of disk space Artifactory allows you to limit the storage space allocated for your repositories.

In the **Admin** module, under **Advanced | Maintenance**, set **Enable Quota Control**, and specify **Storage Space Limit** to specify the percentage of disk space that you allocate for your repositories. An attempt to store binaries above the allocated storage percentage will fail with an error. You may also set **Storage Space Warning** to specify at what percentage of disk space usage to receive a warning from Artifactory.

Storage Quota

<input checked="" type="checkbox"/> Enable Quota Control	
Storage Space Limit (Percentage)*	Storage Space Warning (Percentage)*
<input type="text" value="95"/>	<input type="text" value="85"/>

Limiting the Number of Snapshots

Working with snapshots is a standard development practice, however depending on the number of snapshots that are saved, this can use up large quantities of disk space.

To specify the maximum number of snapshots that may be stored, select the **Repositories** module and click the repository whose settings you want to edit.

In the **Basic** settings, check **Handle Snapshots** and then set the **Max Unique Snapshots** field. This value is zero by default, which means that all snapshots are saved.

Maven Settings

Checksum Policy

Verify against client checksums

Maven Snapshot Version Behavior

Unique

Max Unique Snapshots

5

Handle Releases

Handle Snapshots

Suppress POM Consistency

To avoid issues of concurrency, Artifactory requires that you store a minimum of 2 unique snapshots, however can control the maximum number of snapshots that are stored.

Redundant snapshots are not deleted immediately

Every time you deploy a snapshot, Artifactory will check the value **Max Unique Snapshots** for the repository, and if exceeded will mark any excess old snapshots for deletion. Then, every 5 minutes, Artifactory runs a background process that deletes those oldest snapshots that have been marked. For example, if you set **Max Unique Snapshots** to 5 and deploy a sixth and seventh snapshot to the repository, then next time the background process runs, it will delete the two oldest snapshots.

Deleting Unused Cached Artifacts

When working with [remote repositories](#), to optimize performance, Artifactory locally caches and aggregates snapshots of remote artifacts that are being used. However, if at some point, these artifacts are no longer used, Artifactory can identify and remove them.

You can control how long an unused artifact will remain cached before it is eligible for cleanup. In the **Edit Repository** screen under **Advanced Settings**, specify the number of hours in the **Unused Artifacts Cleanup Period** field.

By default this value is set to zero which means that an artifacts from the corresponding repository are never removed from the cache.

Cache

Unused Artifacts Cleanup Period

Retrieval Cache Period

Assumed Offline Period

Missed Retrieval Cache Period

Cleaning up unused cached artifacts can be scheduled to run automatically during times when demands on the system are low using a Cron expression in the **Admin** module under **Advanced | Maintenance**. You can also invoke an immediate run by clicking "Run Unused Cached Artifacts Cleanup"

Cleanup Unused Cached Artifacts

Cron Expression*

Next Garbage Collection Time

[Run Unused Cached Artifacts Cleanup](#)

Recommended Frequency for Deleting Unused Cached Artifacts

Deleting unused cached artifacts is a resource-intensive operation, so to avoid concurrency and performance issues it is recommended to do it no more than once or twice a day, and preferably during "quiet time" such as outside of regular working hours.

Deleting Complete Versions

Artifactory supports a complete manual deletion of an installed version. This is fully described in [Deleting a Version](#).

User Plugins

Artifactory supports cleanup by allowing you to write custom [User Plugins](#) which you can develop to meet your own specific cleanup requirements.

JFrog provides a number of [cleanup scripts on GitHub](#) which you can use as provided or modify to suit your own needs. For example the following [artifactCleanup](#) plugin deletes artifacts that have not been downloaded for a specified number of months.

Manual Cleanup with the REST API

Using the Artifactory [REST API](#), you may write scripts to implement virtually any custom cleanup logic. This provides you with an extensive and flexible set of customization capabilities as provided by the REST API.

Examples:

- Use the REST API as described [Artifacts Not Downloaded Since](#), to identify artifacts that have not been downloaded since a specific Java epoch, and then remove them.
- Use the REST API as described in [Artifacts Created in Date Range](#) to identify artifacts created within a specific date range and then remove them.

Discarding old builds with Jenkins Artifactory plugin

When using Jenkins for continuous integration, you can configure a policy to discard old builds that are stored in Artifactory along with their artifacts.

For more details please refer to the [Artifactory Plugin](#) page of the [Jenkins Wiki Documentation](#).

Getting Support

Overview

JFrog provides SLA based support for Pro and Enterprise licensing tiers. If you have purchased one of these tiers you may contact JFrog support through the JFrog Support Portal. In most cases, JFrog support will require some initial information about your system and relevant log files. In order to expedite handling of your issue, Artifactory lets you generate all the initially required information in the **Admin** module **Support Zone** screen. When opening a support ticket, you can attach the information bundle to expedite handling of your issue.

Artifactory OSS and Pro users

If you are running Artifactory on an OSS license, and therefore do not have access to JFrog Support Portal, you may visit [JFrog website support page](#) to access the Artifactory Community Forum.

Availability

Support Zone is only available for Artifactory on-prem installations.

Page Contents

- [Overview](#)
- [Requesting Support](#)
- [Collecting an Information Bundle](#)
- [Previously Created Bundles](#)
- [REST API](#)

Requesting Support

To request support, create an [information bundle](#) with the relevant information, and then login to [JFrog Support Portal](#) where you can open a support ticket and attach the information bundle.

What should I include?

Unless you are sure about the information JFrog support will need in order to address your issue, we recommend providing all items in the information bundle you upload.

Collecting an Information Bundle

The support zone provides a variety of options to select what information is included in the bundle you provide JFrog support.

The screenshot shows the 'Support Zone' interface in JFrog Artifactory. The page title is 'Support Zone'. Below the title is a section titled 'Information To Collect'. A message states: 'The support info bundle is not sent to JFrog support directly. Once you completed the download log in to JFrog Support Portal and open a relevant ticket.' Below this message are several checkboxes, all of which are checked: 'System Info', 'Security Descriptor', 'Config Descriptor', 'Configuration Files', 'Storage Summary', 'Scrub Passwords and Private Information', and 'Thread Dump'. There are two input fields: 'Number of Thread Dumps' with the value '1' and 'Interval (Milliseconds)' with the value '0'. There is also a 'System Logs' checkbox which is checked, and a 'Date Span' dropdown menu set to 'Last 24 Hours'. A green 'Create' button is located at the bottom right of the form.

System info	If checked, provides information about your system including storage, system properties, JVM information and plugin status. For details please refer to System Information .
Security descriptor	If checked, provides information about how you have security configured in Artifactory. For details please refer to Security Configuration Descriptor .
Config descriptor	If checked, provide your Artifactory config descriptor which includes detailed information on how Artifactory and its repositories are configured. For details please refer to Global Configuration Descriptor .
Configuration files	If checked, provides configuration files that affect Artifactory's functionality.
Storage summary	If checked, provides information about your system's storage including binaries, file store, and repositories. For details, please refer to Monitoring Storage .
Scrub passwords and private information	If checked, passwords and private information such as email addresses are removed from all items in the information bundle.
Thread dump	If checked, Artifactory will create a thread dump for all running threads. By default a single thread dump is created, however, to get a picture of how data may change over time, you can request several thread dumps separated by a specified time interval with the Number of Thread Dumps and Interval fields.

System logs	<p>If checked, system logs are included in the information bundle. You may specify the time span for which system logs should be included.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>Date range Date range considers files according to the time stamp present in the file name, not by its contents.</p> </div>
--------------------	--

Once you have checked all the information items you wish to include in your information bundle, click "Create" to create the bundle.

Artifactory HA
When creating an information bundle for an Artifactory HA installation, the bundle is created by the specific HA node that happens to handle the "Create" request.

Resource intensive operations
Note that creating a **Thread dump** and **System logs** may be resource intensive operations and may create large information bundles.

Previously Created Bundles

Every information bundle you create is stored in Artifactory and is available for download under **Previously Created Bundles**.

REST API

Artifactory REST API provides the following endpoints you can use to work with information bundles:

Create Bundle	Create a new support information bundle
List Bundles	Lists previously created bundle currently stored in the system
Get Bundle	Downloads a previously created bundle currently stored in the system
Delete Bundle	Deletes a previously created bundle from the system.

Artifactory High Availability

Overview

From version 3.1, Artifactory supports a High Availability network configuration with a cluster of 2 or more, active/active, read/write Artifactory servers on the same Local Area Network (LAN).

Setting up several servers in an HA configuration is supported with an [Enterprise License](#) and presents several benefits to your organization:

Maximize Uptime

Artifactory HA redundant network architecture means that there is no single-point-of-failure, and your system can continue to operate as long as at least one of the Artifactory nodes is operational. This maximizes your uptime and can take it to levels of up to "five nines" availability.

Requires an **Enterprise license**

Page Contents

- [Overview](#)
 - [Maximize Uptime](#)
 - [Manage Heavy Loads](#)
 - [Minimize Maintenance Downtime](#)
- [Architecture](#)

Manage Heavy Loads

By using a redundant array of Artifactory server nodes in the network, your system can accommodate larger load bursts with no compromise to performance. With horizontal server scalability, you can easily increase your capacity to meet any load requirements as your organization grows.

Minimize Maintenance Downtime

By using an architecture with multiple Artifactory servers, Artifactory HA lets you perform most maintenance tasks with no system downtime.

Artifactory HA Version

From version 5.0, Artifactory HA has undergone a major change in infrastructure and uses a binary provider that manages the distribution of files across the cluster nodes and supports cloud-native storage providers.

This guide provides instructions for installing and using Artifactory HA from version 5.0 and above.

- Network Topology
 - Load Balancer
 - Artifactory Server Cluster
 - Local Area Network
- Filestore
- Database

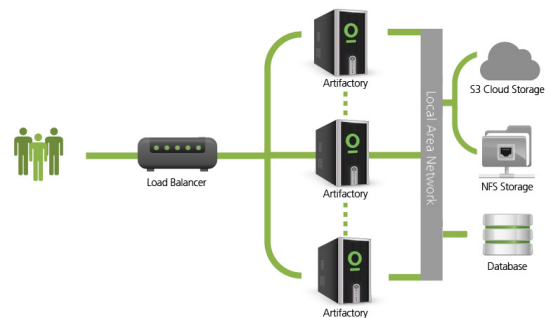
Read more

- [HA Installation and Setup](#)
- [Managing the HA Cluster](#)
- [Migrating Data from NFS](#)
- [Troubleshooting HA](#)

Architecture

Artifactory HA architecture presents a Load Balancer connected to a cluster of two or more Artifactory servers that share a common database where all the Artifactory configuration files are maintained. Binaries may be stored on a Network File System, or using a zoned sharded binary provider as described in [Configuring Sharding for High Availability](#). The Artifactory cluster nodes must be connected through a fast internal LAN in order to support high system performance as well as to stay synchronized and notify each other of actions performed in the system instantaneously. One of the Artifactory cluster nodes is configured to be a "primary" node. Its roles are to execute cluster-wide tasks such as cleaning up unreferenced binaries.

JFrog support team is available to help you configure the Artifactory cluster nodes. It is up to your organization's IT staff to configure your load balancer, database and object store.



Network Topology

Load Balancer

The load balancer is the entry point to your Artifactory HA installation and optimally distributes requests to the Artifactory server nodes in your system. It is the responsibility of your organization to manage and configure it correctly.

Use Artifactory's reverse proxy generator

You may generate configuration snippets for Apache HTTPD and Nginx backed Artifactory High Availability clusters with the built-in [Reverse Proxy generator](#) - it will detect the existing server nodes and add them to the generated configuration file.

The code samples below show some basic examples of load balancer configurations:

▼ [Apache load balancer configuration example...](#)

```
First install the following
modules:
```

```
LoadModule proxy_module
modules/mod_proxy.so
LoadModule
proxy_balancer_module
modules/mod_proxy_balancer.
so
LoadModule
proxy_http_module
modules/mod_proxy_http.so
```

```
Then configure as follows:
```

```
<VirtualHost *:80>
    ServerAdmin
admin@frogs.com
    ServerName
artifactory.jfrog.com
    ServerAlias *.jfrog.com
```

```
<Proxy
balancer://tomcats>
```

```
    # Artifactory server #1
    BalancerMember
http://IP_SERVER_1:PORT
route=art1
```

```
    # Artifactory server #2
    BalancerMember
http://IP_SERVER_1:PORT
route=art2
```

```
    ProxySet
lbmethod=byrequests
</Proxy>
```

```
    ProxyPreserveHost on
    ProxyPass
/balancer-manager !
    ProxyPass /
balancer://tomcats/
    ProxyPassReverse
/artifactory
https://<server
name>/artifactory
    RewriteEngine On
```

```
    RewriteRule    ^/$  
/artifactory      [R,L]
```

```
    LogLevel warn  
    ErrorLog  
/var/log/httpd/apache-hate  
st.error.log  
    CustomLog
```

```
/var/log/httpd/apache-ha-test.access.log combined
</VirtualHost>
```

▼ nginx load balancer configuration example...

```
http {
    ...
    ...
    ...
    upstream artifactory {
        server
        IP_SERVER_1:8081;
        server
        IP_SERVER_2:8081;
    }

    server {
        listen 80;
        server_name
        YOUR_SERVER_NAME;
        ...
        ...
        ...
        rewrite ^/$
        http://$host/artifactory/webapp;
        location / {
            proxy_pass
            http://artifactory;
        }
    }
}
```

More details are available on the [nginx website](#).

Artifactory Server Cluster

Each Artifactory server in the cluster receives requests routed to it by the load balancer. All servers share a common database, and communicate with each other to ensure that they are synchronized on all transactions.

Local Area Network

To ensure good performance and synchronization of the system, all the components of your Artifactory HA installation must be installed on the same high-speed LAN.

In theory, Artifactory HA could work over a Wide Area Network (WAN), however in practice, network latency makes it impractical to achieve the performance required for high availability systems.

Filestore

Artifactory HA offers different options for storing binaries. Some examples are:

- Local file system in which binaries are stored with redundancy using a binary provider which manages synchronizing files between the cluster nodes according to the redundancy defined.
- Cloud storage (currently, Amazon S3 and Google Cloud Storage are supported)
- Network File System (NFS)

Database

Artifactory HA requires an external database, which is fundamental to management of binaries and is also used to store cluster wide configuration files. Currently MySQL, Oracle, MS SQL and PostgreSQL are supported. For details on how to configure any of these databases please refer to [Configuring the Database](#).

Since Artifactory HA contains multiple Artifactory cluster nodes, your database must be powerful enough to service all the nodes in the system. Moreover, your database must be able to support the maximum number of connections possible from all the Artifactory cluster nodes in your system.

If you are replicating your database you must ensure that at any given point in time all nodes see a consistent view of the database, regardless of which specific database instance they access. Eventual consistency, and write-behind database synchronization is not supported.

HA Installation and Setup

Overview

This page describes how to set up a set of Artifactory nodes as an Artifactory HA cluster.

Each of the HA components is configured individually and a common setup file is configured to bring together all of the components in the system as a whole.

Requirements

Version

Artifactory HA is supported from Artifactory 3.1 and above. If you are running a previous version of Artifactory, you first need to upgrade as described in [Upgrading Artifactory](#).

All nodes within the same Artifactory HA installation must be running the same Artifactory version and the same JVM version.

Licensing

Artifactory HA is supported with an [Enterprise License](#). Each node in the cluster must be activated with a different license, however, this is transparently and automatically managed by the Artifactory [Cluster License Manager](#).

Hardware

Artifactory HA requires the following hardware:

- Load balancer
- External database server with a single URL to the database

Network

- All the Artifactory HA components (Artifactory cluster nodes, database server and load balancer) must be within the same fast LAN
- All the HA nodes must communicate with each other through dedicated TCP ports
- Network communications between the cluster nodes must be enabled for each of the cluster nodes.

Database

Artifactory HA requires an external database and currently supports Oracle, MySQL, MS SQL and PostgreSQL. For details on how to configure any of these databases please refer to [Configuring the Database](#).

 Requires an **Enterprise license**

Page Contents

- [Overview](#)
- [Requirements](#)
- [Home Directory](#)
- [Installing Artifactory HA](#)
 - [The Bootstrap Bundle](#)
 - [The Installation Process](#)
 - [Setting Up Your Storage Configuration](#)
 - [Using Filesystem Storage with the NFS](#)
 - [Using Filesystem Storage Without the NFS](#)
 - [Using Cloud Storage With the NFS](#)
 - [Using Cloud Storage Without the NFS](#)
- [Installing the Cluster Nodes](#)
 - [Installing the Primary Node](#)
 - [Creating the](#)

- Bootstrap Bundle
 - Add Licenses
 - Set the URL Base
 - Add Secondary Nodes
- Upgrading Artifactory HA
- Testing Your HA Configuration
- Cluster License Management
 - Adding Licenses
 - License Expiry
 - Deleting Licenses
 - REST API
- Screencast

Home Directory

When setting up Artifactory HA you need to configure the `$ARTIFACTORY_HOME` directory separately for each of the Artifactory cluster nodes in your system.

The general layout of these directories is as follows:

▼ [Click for directory layout...](#)

```

|- $ARTIFACTORY_HOME

  |- access

  |- etc/

    |- ha-node.properties
    |- logback.xml
    |- artifactory.cluster.lic
    |- mimetypes.xml
    |- cluster.id

    |- binarystore.xml
    |- db.properties
    |- plugins
    |- ui
    |- security
    |- access
    |-etc

  |- data/
    |- tmp/
    |- artifactory.properties

  |- logs/

  |- webapps/

  |- tomcat/
  
```

```
|- lib/
    |- <jdbc driver>/

|- bin/

|- misc/

|-backup/

|-support
```

Installing Artifactory HA

An Artifactory HA node is first installed as an Artifactory Pro instance, and is then modified to operate as a node in the HA cluster by configuring the `ARTIFACTORY_HOME/etc/ha-node.properties` file. Once the primary node is set up, it is used to create a bootstrap bundle which is then used to configure the secondary nodes in the cluster.

The Bootstrap Bundle

The bootstrap bundle, `bootstrap.bundle.tar.gz`, contains a set of security keys and configuration files required for the proper functioning of the cluster. During the process of installing and configuring the HA nodes, the bootstrap bundle is generated by calling the [Create Bootstrap Bundle REST API](#) endpoint on the primary node. The same bootstrap bundle should be copied manually to each secondary node during its installation process (into the etc folder). There's no need to unpack the archive, Artifactory handles this process when starting up.

The Installation Process

The binary storage in an HA installation must be accessible to all nodes in the cluster. This is achieved either by mounting a Network File System (NFS) on each cluster node, using shared object storage, or by using the nodes' local file systems while using a mechanism that synchronizes the binaries between them.

The installation procedure involves two stages:

1. **Setting up your storage configuration**

The storage configuration varies depending on your decision for two parameters of your setup:

- a. **Binary store:** Do you plan to use **Filesystem Storage** to store binaries on your nodes' filesystems, or a **Cloud Storage** provider such as S3, GCS or any other S3-compliant provider?
- b. **NFS:** Do you plan to use the Network File System (NFS) or not?

2. **Installing the cluster nodes**

Once your storage is configured and set up, the rest of the installation process is identical

Setting Up Your Storage Configuration

Your choice for binary store and use of NFS or not leads to one of the following four options for setting up your storage configuration:

Using Filesystem Storage with the NFS

▼ [Click here to expand for details...](#)

To set up your HA cluster to use filesystem storage with the NFS, follow these steps which are detailed below:

- Create and configure `ARTIFACTORY_HOME/etc/ha-node.properties`
- Create an NFS mount
- Configure the `binarystore.xml` file

Once you have completed configuring your filestore, you are ready to complete the HA installation process by [installing the cluster nodes](#).

Create ha-node.properties

Create the `ARTIFACTORY_HOME/etc/ha-node.properties` file and populate it with the following parameters:

node.id	<p>Unique descriptive name of this server.</p> <div style="border: 1px solid #f0e68c; padding: 5px; margin: 10px 0;"> <p>Uniqueness Make sure that each node has an id that is unique on your whole network.</p> </div>																		
context.url	<p>The context url that should be used to communicate with this server within the cluster.</p> <p>There are two ways to specify the context.url field:</p> <ul style="list-style-type: none"> • As an explicit IP address • As a host name. In this case, you need to specify the <code>hazelcast.interface</code> field with wildcards. For details, please refer to the description for <code>hazelcast.interface</code> field below. 																		
membership.port	<p>The port that should be used to communicate with this server within the cluster.</p> <p>If not specified, Artifactory will allocate a port automatically, however we recommend to set this to a fixed value to ensure that the port allocated is open to all of your organizations security systems such as firewalls etc.</p>																		
primary	<p>(true false) Indicates if this is the primary server. There must be one (and only one) server configured in the cluster to be the primary server. For other servers this parameter is optional and its value defaults to "false".</p>																		
artifactory.ha.data.dir	<p>This property provides the full path to the root directory of your NFS binary storage.</p>																		
artifactory.ha.backup.dir	<p>This property provides the full path to the root directory of your Artifactory back-up data on the NFS.</p>																		
hazelcast.interface	<p>[Optional] When nodes in the same cluster are running on different networks (e.g. nodes on different docker hosts), set this value to match the server's internal IP address.</p> <p>If you have specified the <code>context.url</code> as a host name, you need to use the wildcard character (i.e., an asterisk - <code>*</code>) so as to include the server's internal IP address as well as that of all members in the cluster.</p> <p>For example, if you have two nodes with the following parameters:</p> <table border="1" style="margin: 10px 0;"> <thead> <tr> <th>Node</th> <th>IP</th> <th>Host name</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>10.1.2.22</td> <td>node.a</td> </tr> <tr> <td>B</td> <td>10.1.3.33</td> <td>node.b</td> </tr> </tbody> </table> <p>then the <code>hazelcast.interface</code> field should be set to 10.1.*.*</p> <p>Another example, if you have two nodes with the following parameters:</p> <table border="1" style="margin: 10px 0;"> <thead> <tr> <th>Node</th> <th>IP</th> <th>Host name</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>10.1.2.22</td> <td>node.a</td> </tr> <tr> <td>B</td> <td>10.1.2.33</td> <td>node.b</td> </tr> </tbody> </table> <p>then the <code>hazelcast.interface</code> field should be set to 10.1.2.*</p>	Node	IP	Host name	A	10.1.2.22	node.a	B	10.1.3.33	node.b	Node	IP	Host name	A	10.1.2.22	node.a	B	10.1.2.33	node.b
Node	IP	Host name																	
A	10.1.2.22	node.a																	
B	10.1.3.33	node.b																	
Node	IP	Host name																	
A	10.1.2.22	node.a																	
B	10.1.2.33	node.b																	

ha-node.properties file permissions
On Linux, once the `ha-node.properties` file is created, the Artifactory user should be set as its owner and its permissions should be set to 644(-rw-r--r--)

The example below shows how an `ha-node.properties` file may be configured for using filesystem storage with the NFS


```
node.id=art1
context.url=http://10.0.0.121:8081/artifactory
membership.port=10001
primary=true
artifactory.ha.data.dir=/mnt/shared/artifactory/ha-data
artifactory.ha.backup.dir=/mnt/shared/artifactory/ha-backup
hazelcast.interface=192.168.0.2 (optional)
```

Escaping the backslash in Windows systems

Note that in Windows-based system the backslash characters in the paths to the ha-data and ha-backup directories need to be escaped with another backslash. For example:

```
artifactory.ha.data.dir = \\windows\UNC\path\ha-data artifactory.ha.backup.dir = \\windows\UNC\path\ha-backup
```

Create an NFS mount

When setting up Artifactory HA you need to configure the `$ARTIFACTORY_HOME` directory separately for each of the Artifactory cluster nodes in your system, and a common `$DATA_DIR` that is accessible to all nodes to host all your filestore binaries

Create an NFS mount which will be accessible to all nodes. This mount will serve as the `$DATA_DIR`.

In addition, you need to set up a `$BACKUP_DIR` that must be accessible by the master node. It may be located on the same NFS mount, however this is not compulsory.

Privileges

Each of the Artifactory cluster nodes must have full write privileges on the `$DATA_DIR` directory tree and the UID/GID for the artifactory user must match on all nodes.

Mounting the NFS from Artifactory HA nodes

When mounting the NFS on the client side, make sure to add the following option for the `mount` command:

```
lookupcache=none
```

This ensures that nodes in your HA cluster will immediately see any changes to the NFS made by other nodes.

Configure the `binarystore.xml` File

The default `binarystore.xml` that comes with Artifactory out-of-the-box contains the [file-system template](#). Since this is exactly the configuration you need, there is no need to modify the `binarystore.xml` file.

In this configuration, Artifactory uses the `artifactory.ha.data.dir` as the location for all binaries.

You are now ready to complete the HA installation process by [installing the cluster nodes](#).

Using Filesystem Storage Without the NFS

✓ [Click here to expand for details...](#)

To set up your HA cluster to use filesystem storage without the NFS, follow these steps which are detailed below:

- Create and configure `$ARTIFACTORY_HOME/etc/ha-node.properties`
- Configure the `binarystore.xml` file

Create `ha-node.properties`

Create the `$ARTIFACTORY_HOME/etc/ha-node.properties` file and populate it with the following parameters:

node.id	Unique descriptive name of this server. <div style="border: 1px solid orange; padding: 5px;">Uniqueness Make sure that each node has an id that is unique on your whole network.</div>
----------------	--

context.url	<p>The context url that should be used to communicate with this server within the cluster.</p> <p>There are two ways to specify the context.url field:</p> <ul style="list-style-type: none"> • As an explicit IP address • As a host name. In this case, you need to specify the <code>hazelcast.interface</code> field with wildcards. For details, please refer to the description for <code>hazelcast.interface</code> field below. 																		
membership.port	<p>The port that should be used to communicate with this server within the cluster.</p> <p>If not specified, Artifactory will allocate a port automatically, however we recommend to set this to a fixed value to ensure that the port allocated is open to all of your organizations security systems such as firewalls etc.</p>																		
primary	<p>(true false) Indicates if this is the primary server. There must be one (and only one) server configured in the cluster to be the primary server. For other servers this parameter is optional and its value defaults to "false".</p>																		
hazelcast.interface	<p>[Optional] When nodes in the same cluster are running on different networks (e.g. nodes on different docker hosts), set this value to match the server's internal IP address.</p> <p>If you have specified the <code>context.url</code> as a host name, you need to use the wildcard character (i.e., an asterisk - '*') so as to include the server's internal IP address as well as that of all members in the cluster.</p> <p>For example, if you have two nodes with the following parameters:</p> <table border="1"> <thead> <tr> <th>Node</th> <th>IP</th> <th>Host name</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>10.1.2.22</td> <td>node.a</td> </tr> <tr> <td>B</td> <td>10.1.3.33</td> <td>node.b</td> </tr> </tbody> </table> <p>then the <code>hazelcast.interface</code> field should be set to 10.1.*.*</p> <p>Another example, if you have two nodes with the following parameters:</p> <table border="1"> <thead> <tr> <th>Node</th> <th>IP</th> <th>Host name</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>10.1.2.22</td> <td>node.a</td> </tr> <tr> <td>B</td> <td>10.1.2.33</td> <td>node.b</td> </tr> </tbody> </table> <p>then the <code>hazelcast.interface</code> field should be set to 10.1.2.*</p>	Node	IP	Host name	A	10.1.2.22	node.a	B	10.1.3.33	node.b	Node	IP	Host name	A	10.1.2.22	node.a	B	10.1.2.33	node.b
Node	IP	Host name																	
A	10.1.2.22	node.a																	
B	10.1.3.33	node.b																	
Node	IP	Host name																	
A	10.1.2.22	node.a																	
B	10.1.2.33	node.b																	

ha-node.properties file permissions

On Linux, once the `ha-node.properties` file is created, the Artifactory user should be set as its owner and its permissions should be set to 644(-rw-r--r--)

The example below shows how the `ha-node.properties` file might be configured for your cluster nodes to use filesystem storage without the NFS:

```
node.id=art1
context.url=http://10.0.0.121:8081/artifactory
membership.port=10001
primary=true
hazelcast.interface=192.168.0.2 (optional)
```

Configure the binarystore.xml File

The default `binarystore.xml` that comes with Artifactory out-of-the-box contains the file-system template which uses the NFS. Therefore, to setup your filestore so that it doesn't use the NFS, you need to modify this file.

Take care when modifying binarystore.xml

Making changes to this file may result in losing binaries stored in Artifactory!

If you are not sure of what you are doing, please contact JFrog Support for assistance.

We recommend using the [cluster-file-system](#) template which is one of the built-in templates that come with Artifactory out-of-the-box. This configuration uses the default filestore location (under `$ARTIFACTORY_HOME/data`) to store binaries locally on the filesystem, unless specified otherwise. A mechanism connected to all other nodes in the cluster is used to keep binaries synchronized and accessible to all nodes, based on the required redundancy value (which is 2 by default).

How to use the cluster-file-system template

To learn how to configure your `binarystore.xml` to use the cluster-file-system template, please refer to [Basic Configuration Elements](#) under [Configuring the Filestore](#).

If your cluster has only two nodes, we recommend modifying the `lenientLimit` from its default value of 0 which would prevent writes to Artifactory if one of the nodes goes down.

You are now ready to complete the HA installation process by [installing the cluster nodes](#).

Using Cloud Storage With the NFS

✓ [Click here to expand for details...](#)

To set up your HA cluster to use cloud storage with the NFS, follow these steps which are detailed below:

- Create and configure `$ARTIFACTORY_HOME/etc/ha-node.properties`
- Create an NFS mount
- Configure the `binarystore.xml` file

Create ha-node.properties

Create the `ha-node.properties` file and populate it with the following parameters:

node.id	<p>Unique descriptive name of this server.</p> <div style="border: 1px solid #f0e68c; padding: 5px; margin-top: 10px;"> <p>Uniqueness Make sure that each node has an id that is unique on your whole network.</p> </div>
context.url	<p>The context url that should be used to communicate with this server within the cluster.</p> <p>There are two ways to specify the <code>context.url</code> field:</p> <ul style="list-style-type: none"> • As an explicit IP address • As a host name. In this case, you need to specify the <code>hazelcast.interface</code> field with wildcards. For details, please refer to the description for <code>hazelcast.interface</code> field below.
membership.port	<p>The port that should be used to communicate with this server within the cluster.</p> <p>If not specified, Artifactory will allocate a port automatically, however we recommend to set this to a fixed value to ensure that the port allocated is open to all of your organizations security systems such as firewalls etc.</p>
primary	<p>(true false) Indicates if this is the primary server. There must be one (and only one) server configured in the cluster to be the primary server. For other servers this parameter is optional and its value defaults to "false".</p>
artifactory.ha.data.dir	<p>This property provides the full path to the root directory of your NFS binary storage.</p>
artifactory.ha.backup.dir	<p>This property provides the full path to the root directory of your Artifactory back-up data on the NFS.</p>

hazelcast.interface

[Optional] When nodes in the same cluster are running on different networks (e.g. nodes on different docker hosts), set this value to match the server's internal IP address.

If you have specified the `context.url` as a host name, you need to use the wildcard character (i.e., an asterisk - `*`) so as to include the server's internal IP address as well as that of all members in the cluster.

For example, if you have two nodes with the following parameters:

Node	IP	Host name
A	10.1.2.22	node.a
B	10.1.3.33	node.b

then the `hazelcast.interface` field should be set to `10.1.*.*`

Another example, if you have two nodes with the following parameters:

Node	IP	Host name
A	10.1.2.22	node.a
B	10.1.2.33	node.b

then the `hazelcast.interface` field should be set to `10.1.2.*`

The example below shows how the `ha-node.properties` file might be configured for your cluster nodes to use cloud storage with the NFS:

```
node.id=art1
context.url=http://10.0.0.121:8081/artifactory
membership.port=10001
primary=true
artifactory.ha.data.dir=/mnt/shared/artifactory/ha-data
artifactory.ha.backup.dir=/mnt/shared/artifactory/ha-backup
hazelcast.interface=192.168.0.2
```

Escaping the backslash in Windows systems

Note that in Windows-based system the backslash characters in the paths to the `ha-data` and `ha-backup` directories need to be escaped with another backslash. For example:

```
artifactory.ha.data.dir = \\windows\UNC\path\ha-data
artifactory.ha.backup.dir = \\windows\UNC\path\ha-backup
```

ha-node.properties file permissions

On Linux, once the `ha-node.properties` file is created, the `Artifactory` user should be set as its owner and its permissions should be set to `644(-rw-r--r--)`

Create an NFS mount

When setting up Artifactory HA you need to configure the `$ARTIFACTORY_HOME` directory separately for each of the Artifactory cluster nodes in your system, and a common `$DATA_DIR` that is accessible to all nodes to host all your filestore binaries

Create an NFS mount which will be accessible to all nodes. This mount will serve as the `$DATA_DIR`.

In addition, you need to set up a `$BACKUP_DIR` that must be accessible by the master node. It may be located on the same NFS mount, however this is not compulsory.

Privileges

Each of the Artifactory cluster nodes must have full write privileges on the `$DATA_DIR` directory tree.

Mounting the NFS from Artifactory HA nodes

When mounting the NFS on the client side, make sure to add the following option for the `mount` command:

```
lookupcache=none
```

This ensures that nodes in your HA cluster will immediately see any changes to the NFS made by other nodes.

Configure the `binarystore.xml` file

The default `binarystore.xml` that comes with Artifactory out-of-the-box contains the [file-system template](#). Therefore, to setup your filestore so to use cloud storage with the NFS, you need to modify this file.

Warning: Take care when modifying the `binarystore.xml` file

Making changes to this file may result in losing binaries stored in Artifactory!

If you are not sure of what you are doing, please contact JFrog Support for assistance.

We recommend using either the [s3](#) chain or the [google-storage](#) chain which are among the [built-in chain templates](#) that come with Artifactory out-of-the-box. These chains use the shared filestore location (under `$DATA_DIR`) to store binaries in a staging area, before they are moved to the cloud storage.

Tip: To learn how to configure your `binarystore.xml` to use the `s3` and `google-storage` chain templates, please refer to [Basic Configuration Elements](#) under [Configuring the Filestore](#).

You are now ready to complete the HA installation process by [installing the cluster nodes](#).

Using Cloud Storage Without the NFS

✓ [Click here to expand for details...](#)

To set up your HA cluster to use cloud storage without the NFS, follow these steps which are detailed below:

- Create and configure `$ARTIFACTORY_HOME/etc/ha-node.properties`
- Configure the `binarystore.xml` file

Create `ha-node.properties`

Create the `ha-node.properties` file and populate it with the following parameters:

node.id	Unique descriptive name of this server. <div style="border: 1px solid #ffc107; padding: 5px; text-align: center;">Uniqueness Make sure that each node has an id that is unique on your whole network.</div>
context.url	The context url that should be used to communicate with this server within the cluster. There are two ways to specify the <code>context.url</code> field: <ul style="list-style-type: none">• As an explicit IP address• As a host name. In this case, you need to specify the <code>hazelcast.interface</code> field with wildcards. For details, please refer to the description for <code>hazelcast.interface</code> field below.
membership.port	The port that should be used to communicate with this server within the cluster. If not specified, Artifactory will allocate a port automatically, however we recommend to set this to a fixed value to ensure that the port allocated is open to all of your organizations security systems such as firewalls etc.
primary	(true false) Indicates if this is the primary server. There must be one (and only one) server configured in the cluster to be the primary server. For other servers this parameter is optional and its value defaults to "false".

hazelcast.interface

[Optional] When nodes in the same cluster are running on different networks (e.g. nodes on different docker hosts), set this value to match the server's internal IP address.

If you have specified the `context.url` as a host name, you need to use the wildcard character (i.e., an asterisk - `*`) so as to include the server's internal IP address as well as that of all members in the cluster.

For example, if you have two nodes with the following parameters:

Node	IP	Host name
A	10.1.2.22	node.a
B	10.1.3.33	node.b

then the `hazelcast.interface` field should be set to `10.1.*`

Another example, if you have two nodes with the following parameters:

Node	IP	Host name
A	10.1.2.22	node.a
B	10.1.2.33	node.b

then the `hazelcast.interface` field should be set to `10.1.2.*`

ha-node.properties file permissions

On Linux, once the `ha-node.properties` file is created, the Artifactory user should be set as its owner and its permissions should be set to `644(-rw-r--)`

The example below shows how the `ha-node.properties` file might be configured for your cluster nodes to use cloud storage without the NFS:

```
node.id=art1
context.url=http://10.0.0.121:8081/artifactory
membership.port=10001
primary=true
hazelcast.interface=192.168.0.2 (optional)
```

Configure the `binarystore.xml` File

The default `binarystore.xml` that comes with Artifactory out-of-the-box contains the file-system template. Therefore, to setup your filestore so to use cloud storage without the NFS, you need to modify this file.

Take care when modifying `binarystore.xml`

Making changes to this file may result in losing binaries stored in Artifactory!

If you are not sure of what you are doing, please contact JFrog Support for assistance.

We recommend using either the `cluster-s3` chain or the `cluster-google-storage` chain which are among the built-in templates that come with Artifactory out-of-the-box. These templates use a mechanism connected to all other nodes in the cluster to keep binaries synchronized and accessible to all nodes according to the required redundancy (which is 2 by default). Binaries are first stored locally on each node (under `$ARTIFACTORY_HOME/data/eventual` by default), with additional copies on other nodes according to the redundancy configured, before moving on to persistent cloud storage.

How to use the `s3` and `google-storage` chain templates

To learn how to configure your `binarystore.xml` to use the `cluster-s3` and `cluster-google-storage` chain templates, please refer to [Basic Configuration Elements](#) under [Configuring the Filestore](#).

You are now ready to complete the HA installation process by [installing the cluster nodes](#).

Installing the Cluster Nodes

Once you have completed setting up your filestore configuration, the process for installing the cluster nodes is identical and described in the steps below:

1. Install the primary node
2. Create the bootstrap bundle
3. Add licenses
4. Set the cluster's URL Base
5. Add secondary nodes

Installing the Primary Node

Go through a regular installation of Artifactory Pro as described in [Installing Artifactory](#), and then convert it to be the HA primary node by adding the `ha-node.properties` file you created when you set up your storage configuration to the `$ARTIFACTORY_HOME/etc`. Do not start up the instance yet.

Note that an external database must be configured for usage at this point, as mentioned in the [Requirements](#) section

You should also verify that your database JDBC driver is correctly located in `$ARTIFACTORY_HOME/tomcat/lib` for each Artifactory cluster node.

Creating the Bootstrap Bundle

First, start up the primary node. Once your primary node is up and running, you can create the bootstrap bundle by calling the [Create Bootstrap Bundle](#) REST API endpoint on the primary node. This creates the bundle, `bootstrap.bundle.tar.gz`, and stores it under `$ARTIFACTORY_HOME/etc`. You will need the bootstrap bundle later on when adding secondary nodes.

Note: The bootstrap bundle file is only used when none of the files it includes are present in the corresponding locations in the secondary cluster nodes. Once Artifactory is finished with it (either used it or deemed unnecessary) the bundle file is deleted as it contains sensitive files.

Tip: We recommend backing up the bootstrap bundle to a folder that is different from where the Artifactory cluster data or `ARTIFACTORY_HOME` folder are located until you have added all your secondary nodes and have verified that the cluster is up and running correctly.

Add Licenses

There are several ways you can add licenses to the cluster:

- Using the Cluster License Manager UI or REST API as described in [Adding Licenses](#)
- As part of the [onboarding wizard](#) you will get when you start up Artifactory for the first time
- Using the [YAML configuration file](#) (NOTE: this will only work for the primary node)

Since currently, the only operative node is the primary node, you can install your licenses there. Once you add the secondary nodes to the cluster, they will be licensed automatically through the [Cluster License Manager](#).



All licenses used must be Enterprise licenses.

Set the URL Base

After you have installed the node and verified that your system is working correctly as an HA installation, you should configure the **Custom URL Base**.

In the **Admin** tab under **Configuration | General**, set the **Custom URL Base** field to the URL of the Load Balancer.

Add Secondary Nodes

You should also verify that your database JDBC driver is correctly located in `$ARTIFACTORY_HOME/tomcat/lib` for each Artifactory cluster node.

To add secondary nodes, for each node, follow these steps:

1. Create an `ha-node.properties` file according to how you want to [set up your storage configuration](#).
2. Go through a new Artifactory Pro installation as described in [Installing Artifactory](#). **Do not start up the instance yet.** Note that an external database **must** be configured for usage at this point, as mentioned in the [Requirements](#) section
3. Once the Artifactory Pro installation is complete, add the `ha-node.properties` file you created to the `$ARTIFACTORY_HOME/etc` folder.
4. Copy the bootstrap bundle you created on the primary node, `bootstrap.bundle.tar.gz`, to the `$ARTIFACTORY_HOME/etc` folder on the secondary node.

Warning: Bootstrap Bundle and db.properties

This is a critical step in the upgrade process. The bootstrap bundle must be installed in each secondary node before you start it up for it to operate correctly in the cluster.

Note also, if the `$ARTIFACTORY_HOME/etc` folder in your secondary node already contains a `db.properties` file, make sure to remove it. Presence of this file will prevent the bootstrap bundle from being properly extracted when you start up the secondary node causing the upgrade to fail.

5. Start up the cluster node. Upon starting up, the node is automatically allocated a license by the [Cluster License Manager](#), and is automatically configured through the bootstrap bundle.
6. [Test your HA configuration](#) after each cluster node that you add to your system.

Warning: Ensure network communication

Make sure that network communication is enabled **between the cluster nodes** for each of the following:

- context.url
- hazelcast.interface and membership.port (used together. For example, `172.24.0.1:10001`)

Upgrading Artifactory HA

Upgrading Artifactory HA depends on which version you are starting from. For detailed instructions, please refer to [Upgrading an Enterprise HA Cluster](#).

Testing Your HA Configuration

The following are a series of tests you can do to verify that your system is configured correctly as an HA installation:

1. Directly Access the Artifactory UI for the server you have just configured
2. In the **Admin** module go to **Advanced | System Logs** to view the log and verify that you see an entry for **HA Node ID**.

```
Artifactory HA
Version: 5.x-SNAPSHOT
Revision: 526
Artifactory Home: '/home/daniela/artifactory-pro-4.12.2-B'
Artifactory data dir: 'null'
Artifactory backup dir: 'null'
HA Node ID: 'art-b'
```

3. The bottom of the module navigation bar should also indicate that you are running with an Enterprise licens. In case of an error you will see an error message in the page header.



4. Access Artifactory through your load balancer and log in as **Admin**.
5. In the **Admin** module go to **Configuration**. There should be a section called **High Availability**. When selected you should see a table with details on all the Artifactory nodes in your cluster as displayed below.

ID	Start Time	URL	Membership Port	State	Role	Last Heartbeat	Version	Revision	Release Date
ha_artifactory_1_1	22-07-15 10:12:39 UTC	http://172.17.0.19:8081/artifactory	10042	Running	Primary	22-07-15 10:13:19 UTC	4.0.0	1677	22-07-15 09:45:32 UTC
ha_artifactory_1_2	22-07-15 10:13:17 UTC	http://172.17.0.20:8081/artifactory	10042	Running	Member	22-07-15 10:13:21 UTC	4.0.0	1677	22-07-15 09:45:32 UTC
ha_artifactory_1_3	22-07-15 10:13:17 UTC	http://172.17.0.21:8081/artifactory	10042	Running	Member	22-07-15 10:13:20 UTC	4.0.0	1677	22-07-15 09:45:32 UTC

6. In the **Admin** module under **Configuration | General**, verify that the **Custom URL Base** field is correctly configured to the URL of the Load Balancer.

Cluster License Management

Artifactory 5.0 introduces an automated license management interface for HA clusters through which all licenses are allocated automatically to nodes as they are added to the cluster. A batch of licenses can be added through the UI and REST API to any node in a cluster.

A new node starting up will request an available license from the pool automatically, and will be allocated the license with the latest expiry date. The license is also automatically returned to the pool if the node is shut down or removed from the HA cluster.

Which license is allocated?

Note that adding a license through a node does not necessarily mean that the license will be attached to that specific node. The license is added to the pool available and the available license with the latest expiry date will be allocated to the node.

Once you have purchased a set of licenses, they are provided to you as a space-separated or newline-separated list.

Adding Licenses

There are three ways that licenses can be added to an HA cluster:

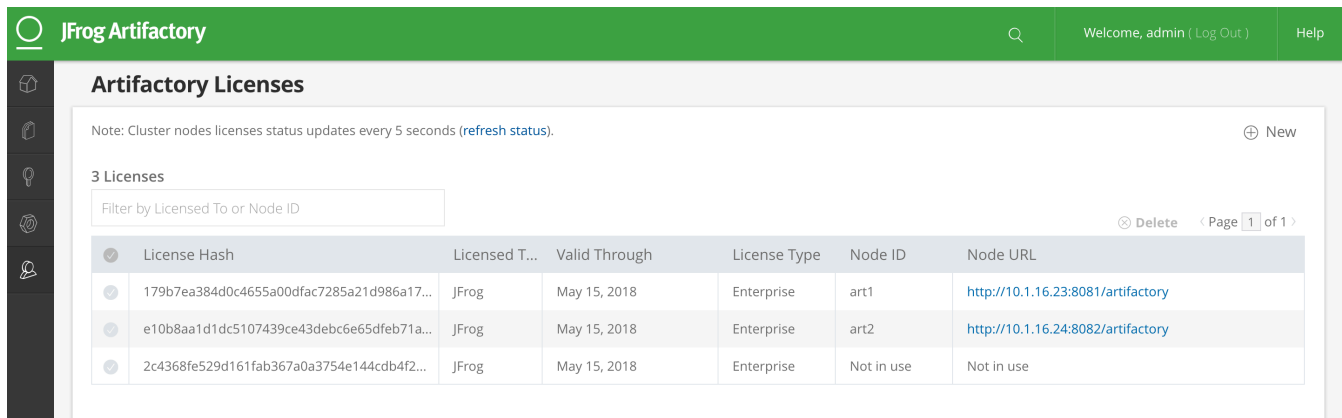
- [Through the UI](#)
- [Using the REST API](#)
- [Adding them to the primary node's filesystem](#) (for automation).

Specifying multiple licenses

When specifying multiple licenses, whether in the Artifactory UI, using the REST API or in the `artifactory.cluster.license` file, make sure that the licenses are separated by a newline.

Using the UI

Through the UI, in the Admin module, under **Configuration | Artifactory Licenses**, you can view all licenses uploaded to your cluster.



The screenshot shows the 'Artifactory Licenses' page in the JFrog Artifactory Admin UI. The page header includes the JFrog logo and 'Artifactory Licenses'. A note at the top states: 'Note: Cluster nodes licenses status updates every 5 seconds (refresh status)'. Below the note, there are 3 licenses listed in a table. The table has columns for License Hash, Licensed To, Valid Through, License Type, Node ID, and Node URL. A 'New' button is located in the top right corner of the table area. The table content is as follows:

License Hash	Licensed To	Valid Through	License Type	Node ID	Node URL
179b7ea384d0c4655a00dfac7285a21d986a17...	JFrog	May 15, 2018	Enterprise	art1	http://10.1.16.23:8081/artifactory
e10b8aa1d1dc5107439ce43debc6e65df71a...	JFrog	May 15, 2018	Enterprise	art2	http://10.1.16.24:8082/artifactory
2c4368fe529d161fab367a0a3754e144cdb4f2...	JFrog	May 15, 2018	Enterprise	Not in use	Not in use

To add licenses to your cluster, click **New** and copy your license key(s) into the **License Key** entry field. You can also simply drag and drop the file containing the license key(s) into the same field. Make sure that each license is separated by a newline.

Add license ✕

Enter your license key below or drop a file [↕](#).

For adding more than one license, use a semicolon or a space as a separator between the different keys.

[Save](#)

Using the REST API

You can also add licenses through the [Install License REST API](#) endpoint

Using the Primary Node's Filesystem

To accommodate spinning up Artifactory HA nodes using automation, **before booting up your primary node**, you can place the `artifactory.cluster.license` file in its `$ARTIFACTORY_HOME/etc` folder. Upon being booted up, the primary node automatically extracts one of the licenses.

Similarly, upon being started up, each secondary node also automatically extracts one of the remaining available licenses.

License Expiry

Nodes running with a license that is about to expire will automatically be updated with a new license available from the pool. Artifactory administrators can manually delete the expired license from within the UI or using REST API.

Deleting Licenses

A license can be deleted under one the following conditions:

- It is not currently being used,
- There is an alternative license available in the pool. In this case, the node to which the deleted license was attached will automatically be allocated with an alternative license.

Perpetual License

Note that Artifactory licenses are perpetual and may continue to activate an Artifactory instance indefinitely, however, an instance running on an expired license may not be upgraded and is not eligible for support.

REST API

You can manage your Artifactory HA licenses using the [HA License Information](#), [Install HA Cluster Licenses](#) and [Delete HA Cluster License REST API](#) endpoints.

Screencast

Managing the HA Cluster

Overview

You can view the status of, and manage your HA cluster nodes in the **Admin** module under **Configuration | High Availability**.

This screen displays a table with details on all the Artifactory nodes in your cluster as displayed below:

Configure High Availability

All Artifactory Nodes

ID	Start Time	URL	Membership Port	State	Role	Last Heartbeat	Version	Revision	Release Date
ha_artifactory_1_1	22-07-15 10:12:39 UTC	http://172.17.0.19:8081/artifactory	10042	Running	Primary	22-07-15 10:13:19 UTC	4.0.0	1677	22-07-15 09:45:32 UTC
ha_artifactory_1_2	22-07-15 10:13:17 UTC	http://172.17.0.20:8081/artifactory	10042	Running	Member	22-07-15 10:13:21 UTC	4.0.0	1677	22-07-15 09:45:32 UTC
ha_artifactory_1_3	22-07-15 10:13:17 UTC	http://172.17.0.21:8081/artifactory	10042	Running	Member	22-07-15 10:13:20 UTC	4.0.0	1677	22-07-15 09:45:32 UTC

Requires an **Enterprise license**

Page Contents

- [Overview](#)
- [Monitoring for Unresponsive Nodes](#)
- [Removing an Unused Node](#)

The table columns are as follows:

ID	Unique descriptive name of the server.
Start Time	The time that the server was started .
URL	The context URL that should be used to communicate with this server within the cluster.
Membership Port	The port that should be used to communicate with this server within the cluster.
State	<p>The current state of the server as follows:</p> <ul style="list-style-type: none"> • Offline - The node has started in an invalid state (For example, same HA license, or the same node id has been set on two different nodes). In this case you should the specific server logs for details. • Starting - The node is starting up. • Running - The node is up and running. This is the normal state for a node. • Stopping - The node is in the process of shutting down. • Stopped - The node is shut down. • Converting - The node is converting database tables and configuration files.
Role	<p>The role of the server as follows:</p> <ul style="list-style-type: none"> • Primary - The primary node. • Member - A regular member node. • Standalone - The node is not configured into your HA cluster. It is running as a separate installation of Artifactory (Pro or Open Source).
Last Heartbeat	The last time this server signaled that it is up and running. By default, each node signals every 5 seconds.
Version	The Artifactory version running on this cluster node.
Revision	The Artifactory revision number running on this cluster node.

Release	The Artifactory release running on this cluster node.
----------------	---

Monitoring for Unresponsive Nodes

You can use the **Last Heartbeat** field to identify unresponsive nodes. If a node abruptly stops working (e.g. system crash on the server), then it may not be able to correctly update its **State** value, and will continue to appear as **Running**.

However, since the server Heartbeat does not get updated for a long interval of time, it is displayed in red with a warning sign as displayed below.

In this case you should check that the corresponding server is up and running and fully connected to your HA cluster and the database.

Configure High Availability

All Artifactory Nodes < page 1 of 1 >

ID	Start Time	URL	Membership Port	State	Role	Last Heartbeat	Version	Revision	Release Date
ha_artifactory_1_1	22-07-15 10:12:39 UTC	http://172.17.0.19:8081/artifactory	10042	Running	Primary	22-07-15 10:15:24 UTC	4.0.0	1677	22-07-15 09:45:32 UTC
ha_artifactory_1_2	22-07-15 10:14:23 UTC	http://172.17.0.20:8081/artifactory	10042	Running	Member	⚠22-07-15 10:14:23 UTC	4.0.0	1677	22-07-15 09:45:32 UTC
ha_artifactory_1_3	22-07-15 10:13:17 UTC	http://172.17.0.21:8081/artifactory	10042	Running	Member	22-07-15 10:15:25 UTC	4.0.0	1677	22-07-15 09:45:32 UTC

Removing an Unused Node

If you remove a node from your cluster, it will still appear in your database and will therefore be displayed in the list of cluster nodes.


To avoid confusion you should remove it from your list of cluster nodes (and detach it from the database) so that it doesn't interfere with normal cluster behavior.

To do so, hover over the corresponding server from the list and click "Delete".

Configure High Availability

All Artifactory Nodes < page 1 of 1 >

ID	Start Time	URL	Membership Port	State	Role	Last Heartbeat	Version	Revision	Release Date
ha_artifactory_1_1	22-07-15 10:12:39 UTC	http://172.17.0.19:8081/artifactory	10042	Running	Primary	22-07-15 10:15:24 UTC	4.0.0	1677	22-07-15 09:45:32 UTC
ha_artifactory_1_2	22-07-15 10:14:23 UTC	http://172.17.0.20:8081/artifactory	10042	Running	Member	⚠22-07-15 10:14:23 UTC	4.0.0	1677	22-07-15 09:45:32 UTC
ha_artifactory_1_3	22-07-15 10:13:17 UTC	http://172.17.0.21:8081/artifactory	10042	Running	Member	22-07-15 10:15:25 UTC	4.0.0	1677	22-07-15 09:45:32 UTC



When to remove a node

The "Remove" button is only available once a node has not signaled a Heartbeat for a "long" time.

We recommend that you only remove a node from your list if it has indeed been removed from your system.

In case of an error in one of the nodes, or if a node is shut down for maintenance, the **Heartbeat** will not be updated and Artifactory will alert you to this as described in the previous section.

In this case there is no need to remove the node from your list. Once you fix the error and restart the server the **Heartbeat** will be updated and the warning will be dismissed.

Migrating Data from NFS

Overview

Previous to version 5.0, an Artifactory HA installation stored binaries and configuration files on an NFS mount. This mount was used by the `$CLUSTER_HOME` folder to synchronize configuration and binary files between the cluster nodes. From version 5.0, you have the option of migrating your binaries to alternative storage which presents the following advantages:

- The filestore can be distributed between the cluster nodes or on a cloud storage provider (S3)
- Limitations of the network (such as file size limits) no longer affect the filestore
- The cluster nodes do not require access to one central location
- Once removed from the NFS, binaries are stored with redundancy in a clustered sharding configuration

This page is designated for users who have upgraded their Artifactory HA installation from version 4.x to version 5.x. During the upgrade process, all configuration files will have been migrated to the database, and will be synchronized and managed there henceforth, however, the data in these installations is still stored on an NFS mount under the `$CLUSTER_HOME/ha-data` folder which leaves you still reliant on the NFS. While you may continue operating in this mode, you also have the option of migrating your data to alternative storage and removing the NFS mount.

Migrating data is optional. NFS is still supported.

While migrating your data from NFS presents the advantages described above, this is optional. Artifactory 5 still supports an HA cluster storing its data on the NFS.

The instructions on this page describe how to move your binary data away from the `$CLUSTER_HOME/ha-data` folder on the NFS mount allowing you to remove the mount altogether. We will cover three main use cases:

Use Case	Initial State	Final State
1	NFS: All data is stored on the NFS	Local FS: All data is stored on each node's local file system
2	NFS Eventual + S3: NFS is used as the Eventual Binary Provider before copying data over to S3 for persistent object store	Local FS Eventual + S3: Each node's local file system is used as the Eventual Binary Provider before copying data over to S3 for persistent object store
3	NFS: All data is stored on the NFS	Local FS Eventual + S3: Each node's local file system is used as the Eventual Binary Provider before copying data over to S3 for persistent object store

For all these use cases, once the data has been migrated, you will be able to completely remove the NFS mount.

Configuring the Migration

Before migrating your data away from the NFS, make sure all nodes in your HA cluster are up and running. Then, to configure migration of your data for the use cases described above, follow the procedure below:

1. [Verify versions](#)
2. [Verify configuration files are synchronized](#)
3. [Edit the `ha-node.properties` file](#)
4. [Copy data to the new location](#)
5. [Configure `binarystore.xml` to match your setup](#)
6. [Test the configuration](#)

Requires an Enterprise license

Page contents

- [Overview](#)
- [Configuring the Migration](#)
 - [Verifying Versions](#)
 - [Verify Configuration Files are Synchronized](#)
 - [Edit the `ha-node.properties` File](#)
 - [Copy Data to the New Location](#)
 - [Use Case 1](#)
 - [Use Case 2](#)
 - [Use Case 3](#)
 - [Configure `binarystore.xml`](#)
 - [Use Case 1](#)
 - [Use Case 2](#)
 - [Testing Your Configuration](#)

Verifying Versions

Before proceeding with transferring your data, you need to verify that all cluster nodes are installed with exactly the same version which must be 5.0 and above. To verify the version running on each node in your HA cluster, in the **Admin** module under **Configuration | High Availability**, check the **Version** column of the table displaying your HA nodes.

Verify Configuration Files are Synchronized

When upgrading your HA cluster from version 4.x to version 5.x, an automatic conversion process synchronizes the configuration files for all the cluster nodes. This replaces the need for the `$CLUSTER_HOME/ha-etc` folder that was used in v4.x. Once you have verified that all nodes are running the same version, you should verify that all configuration files are synchronized between the nodes. For each node, navigate to its `$ARTIFACTORY_HOME/etc` folder and verify the following:

ha-node.properties	Each node should still have this file configured as described in Create ha-node.properties
db.properties	This file was introduced in Artifactory 5.0 and it defines the connection to the database. The password specified in this file is encrypted by the key in the <code>communication.key</code> file. It should be identical in each cluster node.
binarystore.xml	This file opens up the full set of options to configure your binary storage without the NFS. It will contain the binary provider configuration according to how you wish to store your binaries. For each of the use cases described above, you can find the corresponding binary provider configuration under Configure binarystore.xml .
communication.key	This file contains the key used to encrypt and decrypt files that are used to synchronize the cluster nodes. It should be identical on each cluster node.

From version 5.0, Artifactory HA synchronizes configuration files from the primary to all secondary nodes, a change made to one of these files on the primary triggers the mechanism to synchronize the change to the other nodes.

Sync carefully

Since changes on one node are automatically synchronized to the other nodes, take care not to simultaneously modify the same file on two different nodes since changes you make on one node could overwrite the changes you make on the other one.

Edit the ha-node.properties File

Locate the `ha-node.properties` file in each node under the `$ARTIFACTORY_HOME/etc` and comment out or remove the following entries otherwise Artifactory will continue write according to the previous path you have configured to the shared file system.

```
artifactory.ha.data.dir=/var/opt/jfrog/artifactory-ha
artifactory.ha.backup.dir=/var/opt/jfrog/artifactory-backup
```

Copy Data to the New Location

Once you have verified your configuration files are correctly synchronized, you are ready to migrate your data. The sub-sections below describe how to migrate your data for the three use-cases described in the [Overview](#) above.

Use Case 1: NFS Local FS

For this use case, we first need to ensure that there is enough storage available on each node to accommodate the volume of data in my `/data` folder and the desired redundancy. In general, you need to comply with the following formula:

```
Max storage * redundancy < total space available on all nodes
```

For example,

- If you expect the maximum storage in your environment to be **100 TB**

- Your redundancy is **2**
- You have **4 nodes** in your cluster,

Then each node should have at least **50 TB** of storage available.

For a redundancy of N, copy the data from your NFS to N of the nodes in your cluster.

For example, for a redundancy of 2, and assuming you have two nodes named "Node1" and "Node2" respectively, copy the `$CLUSTER_HOME/ha-data` folder to the `$ARTIFACTORY_HOME/data` folder on each of Node1 and Node2.

Optimize distribution of your files

Once you have copied your filestore to each of the N nodes according to the desired redundancy, we recommend invoking the [Optimize System Storage](#) REST API endpoint in order to optimize the storage by balancing its storage amongst all nodes in the cluster.

Use Case 2: NFS Eventual + S3: Local FS Eventual + S3

This use case refers to using S3 as persistent storage, but is equally applicable to other cloud object store providers such as GCS, CEPH, OpenStack and other supported vendors.

In this use case, you only need to ensure that there are **no files in the `eventual` folder of your NFS**. If any files are still there, they should be moved to your cloud storage provider bucket, or to one of the nodes' `eventual` folder.

Use Case 3: NFS Local FS Eventual + S3

Migrating a filestore for a single installation to S3 is normally an [automatic procedure](#) handled by Artifactory, however, in the case of moving an HA filestore from the NFS, the automatic procedure does not work since the folder structure changes.

In this case, you need to copy the data under `$CLUSTER_HOME/ha-data` from your NFS to the bucket on your cloud storage provider (here too, other providers described in Use Case 2 are also supported) while making sure that there are no files left in the `_queue` or `_pre` folders of the eventual binary provider on your node's local file system.

Configure `binarystore.xml`

In this step you need to configure the `binarystore.xml` to match the setup you have selected in the use case. Note that the three use cases above use one of two final configurations:

All data is stored on the cluster node's local filesystem (labelled here as **Local FS**)

The cluster nodes use the cluster node's local filesystem as an eventual binary provider and data is persistently stored on S3 (labelled here as **Local FS Eventual + S3**)

Node downtime required

To modify the `binarystore.xml` file for a node, you first need to gracefully shut down the node, modify the file and then restart the node in order for your new configuration to take effect

Local FS

In this example, all data is stored on the nodes' file systems. For the sake of this example, we will assume that:

- We have 3 nodes
- We want redundancy = 1

To accomplish this setup, you need to:

- Copy the data from the `$CLUSTER_HOME/ha-data` on your NFS to the `$ARTIFACTORY_HOME/data` folder on two of the nodes.
- Once all data has been copied, you need to place the `binarystore.xml` under `$ARTIFACTORY_HOME/etc` of each cluster node.
- Finally, you need to gracefully restart each node for the changes to take effect.

Optimizing the redundant storage

After restarting your system, you can trigger optimization using the REST API so that all three nodes are utilized for redundancy. For details, please refer to [Optimize System Storage](#).

Example

In this use case, the `binarystore.xml` used with the NFS before migration would look like the following if you are using one of the default

file-system template.

```
<config version="1">
  <chain template="file-system"/>
</config>
```

After migrating the data, the new `binarystore.xml` placed on each cluster node you can use the `cluster-file-system` template.

```
<config version="2">
  <chain template="cluster-file-system"/>
</config>
```

While you don't need to configure anything else, this is what the `cluster-file-system` template looks like:

Redundancy leniency

We recommend adding the `lenientLimit` parameter to the below configuration under the `sharding-cluster` provider configuration:

```
<lenientLimit>1</lenientLimit>
```

Without this parameter, Artifactory won't accept artifact deployments while the number of live nodes in your cluster is lower than the specified redundancy.

```
<config version="2">
  <chain> <!--template="cluster-file-system"-->
    <provider id="cache-fs" type="cache-fs">
      <provider id="sharding-cluster" type="sharding-cluster">
        <sub-provider id="state-aware" type="state-aware"/>
        <dynamic-provider id="remote-fs" type="remote"/>
      </provider>
    </provider>
  </chain>

  <provider id="state-aware" type="state-aware">
    <zone>local</zone>
  </provider>

  <!-- Shard dynamic remote provider configuration -->
  <provider id="remote-fs" type="remote">
    <zone>remote</zone>
  </provider>

  <provider id="sharding-cluster" type="sharding-cluster">
    <readBehavior>crossNetworkStrategy</readBehavior>
    <writeBehavior>crossNetworkStrategy</writeBehavior>
    <redundancy>2</redundancy>
    <property name="zones" value="local,remote"/>
  </provider>

</config>
```


Local FS Eventual + S3

In this example, data is temporarily stored on the file system of each node using an Eventual binary provider, and is then passed on to your S3 object storage for persistent storage.

In this use case, the `binarystore.xml` used your NFS for cache and eventual with your object store on S3 before migration will look like the following if you are using the S3 template.

```
<config version="2">
  <chain template="s3" />
</config>
```

After migrating your filestore to S3 (and stopping to use the NFS), your `binarystore.xml` should use the `cluster-s3` template as follows:

```
<config version="2">
  <chain template="cluster-s3" />
</config>
```

The `cluster-s3` template looks like this:

Redundancy leniency

We recommend adding the `lenientLimit` parameter to the below configuration under the `sharding-cluster` provider configuration:

```
<lenientLimit>1</lenientLimit>
```

Without this parameter, Artifactory won't accept artifact deployments while the number of live nodes in your cluster is lower than the specified redundancy.

```

<config version="2">
  <chain> <!--template="cluster-s3"-->
    <provider id="cache-fs-eventual-s3" type="cache-fs">
      <provider id="sharding-cluster-eventual-s3"
type="sharding-cluster">
        <sub-provider id="eventual-cluster-s3"
type="eventual-cluster">
          <provider id="retry-s3" type="retry">
            <provider id="s3" type="s3"/>
          </provider>
        </sub-provider>
        <dynamic-provider id="remote-s3" type="remote"/>
      </provider>
    </provider>
  </chain>

  <provider id="sharding-cluster-eventual-s3" type="sharding-cluster">
    <readBehavior>crossNetworkStrategy</readBehavior>
    <writeBehavior>crossNetworkStrategy</writeBehavior>
    <redundancy>2</redundancy>
    <property name="zones" value="local,remote"/>
  </provider>

  <provider id="remote-s3" type="remote">
    <zone>remote</zone>
  </provider>

  <provider id="eventual-cluster-s3" type="eventual-cluster">
    <zone>local</zone>
  </provider>
  <provider id="s3" type="s3">
    <endpoint>http://s3.amazonaws.com</endpoint>
    <identity>[ENTER IDENTITY HERE]</identity>
    <credential>[ENTER CREDENTIALS HERE]</credential>
    <path>[ENTER PATH HERE]</path>
    <bucketName>[ENTER BUCKET NAME HERE]</bucketName>
  </provider>
</config>

```

Because you must configure the s3 provider with parameters specific to your account (but can leave all others with the recommended values), if you choose to use this template, your *binarystore.xml* configuration file should look like this:

```

<config version="2">

  <chain template="cluster-s3"/>

  <provider id="s3" type="s3">
    <endpoint>http://s3.amazonaws.com</endpoint>
    <identity>[ENTER IDENTITY HERE]</identity>
    <credential>[ENTER CREDENTIALS HERE]</credential>
    <path>[ENTER PATH HERE]</path>
    <bucketName>[ENTER BUCKET NAME HERE]</bucketName>
  </provider>

</config>

```

Testing Your Configuration

To test your configuration you can simply deploy an artifact to Artifactory and then inspect your persistent storage (whether on your node's file system or your cloud provider) and verify that the artifact has been stored correctly.

Troubleshooting HA

Artifactory Does Not Start Up

- There are no log file entries in `$ARTIFACTORY_HOME/logs/artifactory.log`

Cause	Something within your <code>\$ARTIFACTORY_HOME</code> or <code>\$CLUSTER_HOME</code> directory is either not defined, or misconfigured
Resolution	In some cases in which the <code>\$ARTIFACTORY_HOME</code> directory tree is not validly constructed, log file entries are written to <code>\$ARTIFACTORY_HOME/logs/catalina/localhost/<date>/logs</code> . Check the contents of this file to see which specific errors were logged.

- After restarting a cluster node which uses a shared NFS mount, the startup fails

Cause	The <code>\$NFS_MOUNT/ha-etc</code> still contains the bootstrap bundle archive used for the last upgrade. Artifactory tries to redeploy the archive's contents to its respective locations under the <code>\$ARTIFACTORY_HOME</code> and runs into a conflict. The <code>jfrog-access.bootstrap.log</code> shows the following output: <div data-bbox="350 1472 1369 1686" style="border: 1px dashed blue; padding: 10px; margin: 10px 0;"> <pre> [jfrog-access] [INFO] Found bootstrap bundle file: /clusterhome/ha-etc/bootstrap.bundle.tar.gz [jfrog-access] [INFO] Deploying bootstrap bundle file to: /var/opt/jfrog/artifactory </pre> </div>
Resolution	Delete the bootstrap bundle archive from the <code>\$NFS_MOUNT/ha-etc</code> folder. Restart the node and the startup should succeed.

- The log says "Stopping Artifactory start up ,another server running converting process".

Cause	You are upgrading more than one server at a time.
--------------	---

Resolution	When upgrading your system, make sure you complete the upgrade process on one server before starting to upgrade the next one.
-------------------	---

▼ The log says "Stopping Artifactory start up ,another server with different version has been found".

Cause	You are trying to install different versions of Artifactory into the same system.
Resolution	Make sure that all the instances of Artifactory installed in your system are the same version.

▼ The log says "Stopping Artifactory since duplicate node ids have been found in registry. If you restarted this server, make sure to wait at least 30 seconds before re-activating it"

Cause	<p>This may happen in one of two cases:</p> <ol style="list-style-type: none"> 1. Two servers are configured into your system with the same <i>node.id</i> specified in the <i>\$ARTIFACTORY_HOME/etc/ha-node.properties</i> file. 2. You have shut down an Artifactory server and tried to restart it within 30 seconds.
Resolution	<p>Make sure that all servers within your Artifactory HA installation have a unique <i>node.id</i> value.</p> <p>Shut down your server and wait at least 30 seconds before you restart it.</p>

▼ The log says "Node could not join cluster. A Configuration mismatch was detected: Incompatible partition count! expected: 8, found: 1 Node is going to shutdown now!"

Cause	A new node has new Artifactory Hazelcast system property values defined <i>\$ARTIFACTORY_HOME/etc/artifactory.system.properties</i> file and it is trying to join a cluster with nodes with different Hazelcast property values.
Resolution	<p>Make sure that all servers within your Artifactory HA installation have the same values.</p> <p>Shut down your cluster (not rolling restart), start the nodes one by one.</p>

 Requires an **Enterprise license**

Page Contents

- Artifactory Does Not Start Up
- Artifactory Starts Up But Remains Offline
- Artifactory UI login still prompts for credentials after a successful attempt
- Artifactory Starts Up But Not as an HA Installation
- A Cluster Node Does Not Synchronize with Other Nodes
- Upgrading from Version 5.4.5 or Below to Version 5.5 or Above Fails

Artifactory Starts Up But Remains Offline

▼ The log says "Changing Artifactory mode to offline since the server is configured as HA but the license does not exist or is not an HA License"

Cause	Your server does not have a valid HA license installed
Resolution	Install a valid HA license in your server and restart it

▼ The log says "Changing Artifactory mode to offline since the local server is running as HA but found no HA server in registry."

Cause	You are starting an Artifactory server as an HA installation, however you already have an Artifactory Pro (or OSS version) running within the same system.
Resolution	Make sure your system is consistent - either you have a set of Artifactory Pro instances running separately, or you all of your servers are configured as Artifactory HA

▼ The log says "Could not find cluster properties"

Cause	Your <code>\$CLUSTER_HOME/ha-etc/cluster.properties</code> file is not defined.
Resolution	When installing Artifactory HA, you need to manually create a <code>\$CLUSTER_HOME</code> directory and the <code>\$CLUSTER_HOME/ha-etc/cluster.properties</code> file. For details please refer to Configuring the Cluster .

Artifactory UI login still prompts for credentials after a successful attempt

▼ Artifactory version 5.x and above running with Hazelcast allow UI logins made without sticky session/persistence configuration required on the load balancer.

Cause	You have not opened the required Hazelcast ports (Artifactory's nodes synchronized memory component) which are configured under: <code>\$ARTIFACTORY_HOME/etc/ha-node.properties</code> file
Resolution	<ol style="list-style-type: none"> 1. Open the membership ports on your operating system level 2. Ensure communication to the newly opened ports from the membering cluster nodes 3. Verify that the UI login works now (no restart is needed)

Artifactory Starts Up But Not as an HA Installation

▼ Artifactory starts up as an instance of Artifactory Pro

Cause	You have not created a valid <code>\$ARTIFACTORY_HOME/etc/ha-node.properties</code> file
Resolution	<ol style="list-style-type: none"> 1. Shutdown the node 2. Delete the <code>\$ARTIFACTORY_HOME/access</code> folder 3. Delete the <code>\$ARTIFACTORY_HOME/etc/security/access</code> folder 4. Delete <code>\$ARTIFACTORY_HOME/etc/security/communication.key</code> 5. Delete <code>\$ARTIFACTORY_HOME/etc/binarystore.xml</code> 6. Delete <code>\$ARTIFACTORY_HOME/etc/db.properties</code> 7. Delete <code>\$ARTIFACTORY_HOME/etc/cluster.id</code> 8. Copy the bootstrap bundle you created on the primary node, <code>bootstrap.bundle.tar.gz</code>, to the <code>\$ARTIFACTORY_HOME/etc</code> folder on the secondary node. 9. Ensure that the bundle is owned by artifactory user (chown <code>artifactory:artifactory bootstrap.bundle.tar.gz</code>) 10. Create a valid <code>\$ARTIFACTORY_HOME/etc/ha-node.properties</code> file as described in Installing Artifactory HA. 11. Start up the node

A Cluster Node Does Not Synchronize with Other Nodes

▼ [The node does not contain the right bootstrap bundle](#)

Cause	The node was installed without the right bootstrap bundle (<i>bootstrap.bundle.tar.gz</i>) or no bootstrap bundle at all.
Resolution	Install the right bootstrap bundle using the following procedure: <ol style="list-style-type: none">1. Shutdown the node2. Delete the <i>\$ARTIFACTORY_HOME/access</i> folder3. Delete the <i>\$ARTIFACTORY_HOME/etc/security/access</i> folder4. Delete <i>\$ARTIFACTORY_HOME/etc/security/communication.key</i>5. Delete <i>\$ARTIFACTORY_HOME/etc/binarystore.xml</i>6. Delete <i>\$ARTIFACTORY_HOME/etc/db.properties</i>7. Delete <i>\$ARTIFACTORY_HOME/etc/cluster.id</i>8. Copy the bootstrap bundle you created on the primary node, <i>bootstrap.bundle.tar.gz</i>, to the <i>\$ARTIFACTORY_HOME/etc</i> folder on the secondary node.9. Ensure that the bundle is owned by artifactory user (chown artifactory:artifactory <i>bootstrap.bundle.tar.gz</i>)10. Start up the node

Upgrading from Version 5.4.5 or Below to Version 5.5 or Above Fails

▼ [The log says "To upgrade your HA installation to this version, you first need to upgrade to version 5.4.6 which implements changes required to accommodate a database schema change."](#)

Cause	Artifactory 5.5 introduces a change to the database schema. To upgrade to this version or above, the database must first be migrated to the new schema
Resolution	Artifactory 5.4.6 implements a process that performs the required database schema migration. To upgrade to version 5.5 or above from version 5.4.5 or below, you first need to upgrade to version 5.4.6 using the normal upgrade procedure according to your installation type, and then upgrade to your desired version (5.5 or above), also using the normal upgrade procedure .

Xray Integration

Overview

JFrog Xray is a universal binary analysis product that works with Artifactory to analyze software components, and reveal a variety of issues at any stage of the software application lifecycle. By scanning binary components and their metadata, recursively going through dependencies at any level, JFrog Xray provides unprecedented visibility into issues lurking in components anywhere in your organization. As a complementary product to Artifactory, JFrog Xray has access to the wealth of metadata Artifactory stores which, combined with deep recursive scanning, puts Xray in a unique position to analyze the relationships between binary artifacts and provide radical transparency into your component architecture to reveal the impact that an issue in one component has on any other.

For more information about the types of analyses that Xray performs, please refer to [Watches](#) in the JFrog Xray User Guide.

How Does It Work

For Xray to perform its analyses it needs to be connected to an instance of Artifactory in order to access its repositories and metadata. Once connected, Xray can index the artifacts in Artifactory's repositories to efficiently access them for [Scanning or Impact Analysis](#). Since the indexing process is resource intensive, Xray does not automatically analyse all of your repositories; you need to specify which repositories should be indexed. All builds are indexed automatically.

Version Compatibility

JFrog Xray can connect to Artifactory from version 4.0 and above.

Page Contents

- [Overview](#)

- How Does It Work
- Version Compatibility
- Configuring the Integration
 - Connecting to JFrog Xray
 - Specifying Repositories for Analysis
 - Per Repository
 - In Bulk
 - Configuring Download Blocking
 - Indexing Artifacts

Configuring the Integration

Configuring Artifactory to work with JFrog Xray involves the following three main steps:

1. [Connecting Artifactory to JFrog Xray](#)
2. [Specifying repositories](#) whose artifacts should be indexed for analysis by Xray and [configuring download blocking](#)
3. [Indexing artifacts](#)

In addition, JFrog Xray should be properly configured as described in [Configuring Xray](#) in the JFrog Xray User Guide

Connecting to JFrog Xray

The connection between Artifactory and Xray is established by Xray which creates a user with "admin" privileges called **xray** in Artifactory in order to access the data it needs to perform its different analyses and functions.

For details, please refer to [Connecting to Artifactory](#) in the JFrog Xray User Guide.

Specifying Repositories for Analysis

For Xray to analyze the artifacts in your installation efficiently, it first needs to index them in its database. If Xray were to index and analyze **all** of the artifacts in your Artifactory installation, that could cause excessive processing and cluttered component graphs which may obscure the significant components you are really interested in. Therefore, to let you focus on the most important artifacts in your Artifactory installation, Xray will only analyze artifacts from repositories your mark for indexing. There is no need to specify builds; all builds are automatically indexed by Xray.

Repositories marked for indexing by Xray are found in the **Admin** module under **Configuration | JFrog Xray**

The screenshot shows the 'JFrog Xray Integration' configuration page in the JFrog Artifactory Admin console. At the top, there is a green header with the JFrog logo and 'JFrog Artifactory' text. Below the header, the page title is 'JFrog Xray Integration'. A checkbox labeled 'Enable Xray Integration' is checked. Below this, there is a section for '15 Repositories' with a search filter 'Filter by Name'. A table lists the configured repositories with columns for Name, Repository Type, and Package Type. The table contains 15 rows of data. At the bottom right of the table, it says 'Page 1 of 1'.

Name	Repository Type	Package Type
cocopods-local	Local	CocoaPods
debian-local	Local	Debian
docker-v2-local	Local	Docker
ext-release-local	Local	Maven
ext-snapshot-local	Local	Maven
libs-release-local	Local	Maven
libs-snapshot-local	Local	Maven
nuget-local	Local	NuGet
plugins-release-local	Local	Maven
plugins-snapshot-local	Local	Maven
ruby-local	Local	Gems
yum-100	Local	YUM
cocopods-remote	Remote	CocoaPods
jcenter	Remote	Maven
tester	Remote	Bower

To enable analysis of repositories in general, you first need to globally enable Xray by setting the **Enable Xray Integration** checkbox.

Once repositories are marked for analysis, Xray will index (and reindex) their artifacts based on different triggers such as adding, deleting and moving artifacts. Artifacts in all builds are indexed automatically by JFrog Xray and re-indexed each time a new build is created.

There are two ways to specify repositories whose artifacts should be indexed:

1. Per repository
2. In bulk

Per Repository

To specify a specific repository for indexing, in the repository Basic configuration, under **Xray Integration**, check **Enable Indexing in Xray**.

JFrog Xray Integration

Enable indexing in Xray

Block Unscanned Artifacts ?

Block Downloads With Severity Above ?

Major

In Bulk

The Xray Integration screen displays the repositories that have been enabled for indexing. To add more repositories for indexing, click **Add**.

Add Repositories To Xray

Filter...

Available Repositories

- ext-release-local
- ext-snapshot-local
- libs-snapshot-local
- plugins-release-local
- plugins-snapshot-local
- docker-remote
- jcenter

Selected Repositories To Add

- libs-release-local

Cancel Save

From the list of **Available Repositories** select the repositories you wish to add for indexing and click "Save".

Configuring Download Blocking

To prevent potentially harmful artifacts from being used by developers, an administrator can prevent them from being downloaded from Artifactory using the following two settings in the repository Basic configuration, under **Xray Integration**:

Block Unscanned Artifacts	When checked, Artifactory will block download of artifacts from this repository until they have been scanned by JFrog Xray.
Block Downloads With Severity Above	When set, Artifactory will block download of artifacts that have been identified to include an issue with a severity of the degree selected at least.

Once these parameters are set, a [System Watch](#) is created in Xray to detect artifacts that meet the set specifications and block their being downloaded.

Indexing Artifacts

Once JFrog Artifactory and JFrog Xray have been configured to work together, artifacts will be indexed for analysis on an ongoing basis according to different events that happen in Artifactory. To set up the initial database of artifacts Xray, you need to invoke indexing manually. For details, please refer to [Indexing Artifacts](#) in the JFrog Xray User Guide.

Bintray Integration

[Bintray](#) is JFrog's platform for storage and distribution of software libraries on the cloud. It is the new way for developers to publish, download and share software across one unified community around the world. The free, cloud-based platform empowers developers to control and streamline the entire process of making software libraries publicly available, with all the services needed to collaborate, advertise and deploy a new software solution.

Distributing Software Through Bintray

For details on how to use Bintray, please refer to the [Bintray User Guide](#).

Naturally, Artifactory integrates with Bintray in more than one way:

- [Remote Search in Bintray's JCenter repository](#) - the most comprehensive collection of Maven artifacts.
- [Information insight from Bintray on artifacts](#) - package description and latest released version.
- [Pushing artifacts to Bintray through a Distribution Repository](#).
- [Complete continuous delivery stack for selected OSS projects based on oss.jfrog.org and Bintray](#).

Read More

- [Bintray info panel](#)
- [Distribution Repository](#)
- [Deploying Snapshots to oss.jfrog.org](#)

Bintray info panel

Overview

As part of Artifactory's integration with Bintray information about components stored in Bintray is fetched and displayed in the Tree Browser under **Package Information**.

To view Bintray Package Information:

- You need to be logged in
- You need to have the Bintray user and API Key configured in your [Artifactory profile](#).
- The selected file type is supported (e.g., pom, jar, war, ear)

Page Contents

- [Overview](#)

The screenshot displays the Artifactory interface for a specific package. At the top, the package name 'artifactory-papi-2.6.4-javadoc.jar' is shown with a download icon and 'Download' and 'Actions' buttons. Below this is a navigation bar with tabs: 'General' (selected), 'Effective Permissions', 'Properties', 'Watchers', 'Builds', and 'Governance'. The main content area is titled 'Info' and contains a table of package details:

Name:	artifactory-papi-2.6.4-javadoc.jar
Repository Path:	libs-release-local/org/artifactory/artifactory-papi/2.6.4/artifactory-papi-2.6.4-javadoc.jar ⓘ
Module ID:	org.artifactory:artifactory-papi:2.6.4:javadoc
Deployed by:	admin
Size:	363.49 KB
Created:	30-04-15 11:30:59 UTC ⓘ
Last Modified:	27-09-12 21:14:33 UTC
Licenses:	Not Found Add Scan Search Archive License File
Downloaded:	2
Last Downloaded:	15-07-15 14:28:43 UTC
Last Downloaded By:	admin
Watching Since:	04-05-15 15:28:42 UTC

At the bottom, there is a 'Package Information' section, highlighted with a red border, containing:

Name:	org.artifactory:artifactory-papi ⓘ
Latest Version:	3.9.2

Distribution Repository

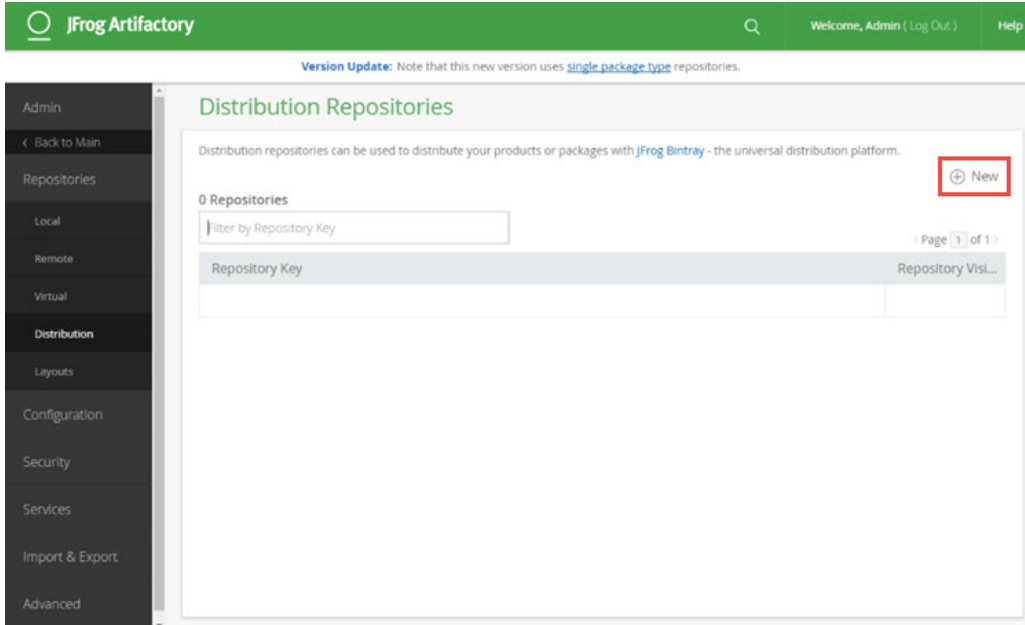
Overview

Artifactory takes its integration with JFrog Bintray to the next step with Distribution Repositories. Distribution repositories provide an easy way to move artifacts from Artifactory to Bintray, for distribution to end users. As

opposed to other repositories in Artifactory, distribution repositories are not typed to a particular package format, but rather, are governed by a set of rules that specify how an artifact that gets to the distribution repository should be routed to its corresponding repository in Bintray.

Configuring a Distribution Repository

You can access your distribution repositories in the **Admin** module under **Repositories | Distribution**.



Page Contents

- Overview
- Configuring a Distribution Repository
 - Connecting to Bintray
 - Basic Distribution Parameters
 - Advanced Settings
- Managing Rules
 - Rule Order
 - Specifying Rule Parameters
 - Repository and Path Filter Parameters
 - Using Unnamed Capture Groups
 - Using Named Capture Groups
 - Enumerating Capture Groups
- Distributing Artifacts
 - Distributing Through the UI
 - Dry Run
 - Distributing via REST API

To set up a new distribution repository, click **New** and execute the following main steps:

1. [Connect to Bintray](#) - obtain authorization from your Bintray account for Artifactory to connect and deploy packages
2. [Configure distribution](#) - specify basic distribution parameters
3. [Configure Advanced Settings](#) - specify advanced settings
4. [Define rules](#) - specifies the rules that govern how this distribution repository will deploy packages to Bintray

Connecting to Bintray

Artifacts are synchronized from your Artifactory distribution repository to Bintray through a Bintray organization to which you have administration privileges. To set up the connection, you first need authorize Artifactory to access your Bintray organization. Artifactory will display a popup dialog where you can enter your Bintray credentials.



JFrog Bintray

Authorize with Bintray

You need to sign into Bintray first

Username

Password

Login

Already logged into Bintray?

If you are already logged into your Bintray account, Artifactory will skip this step

After authorizing access to your account, you need to select the organization in that account that you authorize Artifactory to manage.



JFrog Bintray



JFrog Artifactory

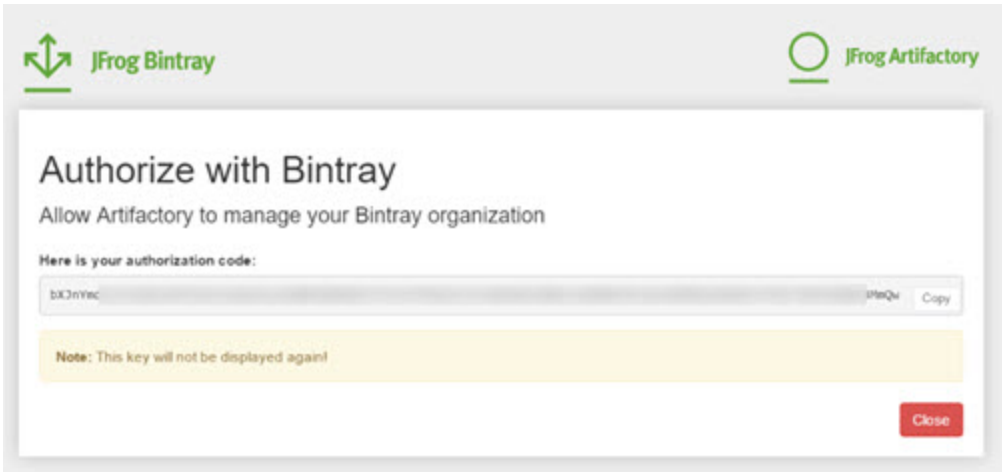
Authorize with Bintray

Allow Artifactory to manage your Bintray organization

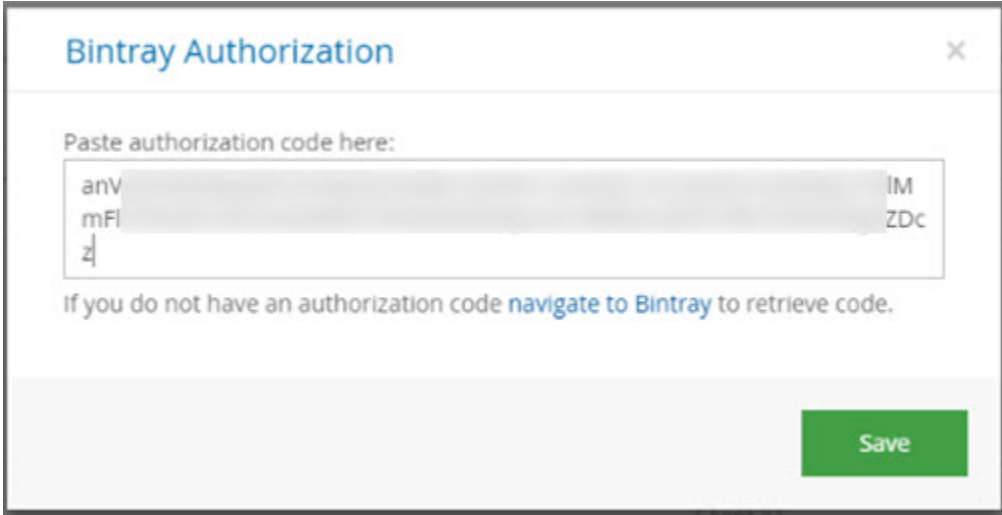
Please select the Bintray organization you wish to authorize

Authorize

Bintray will issue an authorization code which you need to copy, and then paste in your distribution repository configuration.



When you close the popup with the authorization code, Artifactory will display a popup for you to enter it.



Once the process is complete, you can verify that Artifactory has been authorized to access your Bintray organization, in that organization's profile page under OAuth Applications.



Edit systemc's profile
systemc (systemc)

- General
- Members
- Teams
- Products
- Repositories
- GPG Signing
- URL Signing
- Accounts
- OAuth Applications**

OAuth Applications

The following applications are authorized to access your organization:

	Artifactory Client ID: jur4ybdddndaofvrgutdq Permissions: org:systemc:admin	Revoke Access
--	--	-------------------------------

Basic Distribution Parameters

Once you have set up Artifactory as an authorized application in your Bintray organization, you can set up the distribution parameters

New Distribution Repository

Basic Advanced Rules

Repository Key *
distribution-repo

General

Description
Repository for distribution to Bintray

Bintray Application

Client ID
jur4ybdddrrdaofv-vrgutdq

Organization
systemc

Bintray Product

Use This Repository To Distribute a Product ?

Product Name *

Default Repository Settings

Artifactory will use the configuration below to set visibility when creating a new Bintray repository. If distributing to existing repository Artifactory will **not** override the configuration.

Repository Visibility **Private** | Public

Default Package Settings

Artifactory will use the configuration below to set the license and VCS URL when creating a new package. If distributing to existing package in Bintray Artifactory will **not** override the configuration.

Licenses
Select Value...

VCS URL

Cancel < Back Next > **Save & Finish**

Repository Key	The repository key.
General	
Description	A description of the repository.
Bintray Product	
Use This Repository To Distribute a Product	When set, indicates that artifacts distributed through this repository should be linked to a product. Artifactory will create the product (if necessary) and link deployed packages to the product.
Product Name	The product name.
Bintray Application	
customer ID	The client ID assigned by Bintray to Artifactory as an authorized application.
Organization	The Bintray organization which Artifactory is authorized to manage for distribution.

Default Repository Settings	If Artifactory creates a new repository on Bintray for distribution, this setting specifies if the repository should be private or public. If the repository exists, Artifactory will not override its access regardless of this setting.
Default Package Settings	
Licenses	Specifies the OSS licenses that should be attached to any packages distributed through this repository.
VCS URL	Specifies the VCS URL for packages distributed through this repository.

Advanced Settings

Proxy	Select a proxy to use when synchronizing artifacts to Bintray.
GPG Signing	When set, Artifactory will GPG sign artifacts synchronized to Bintray.
GPG Passphrase	The passphrase to use for GPG signing.
Map Properties to Bintray Attributes	Specify a list of properties which, if they annotate the artifact distributed, should be mapped to version attributes in Bintray.

Managing Rules

As opposed to local, remote and virtual repositories in Artifactory, distribution repositories are not limited to a specific package type. Instead, you specify a set of rules, based on package type, and [different filters](#), that give you fine-grained control over how different packages are pushed to Bintray for distribution. To view the rules defined for your repository, click the **Rules** tab. New distribution repositories come with a pre-defined set of rules which you can modify, delete or add to as needed.

New Distribution Repository

Basic > Advanced > **Rules**

+ New

13 Rules

Filter by Name

Delete < Page 1 of 1 >

<input checked="" type="checkbox"/>	Name	Type
<input type="checkbox"/>	Bower-default	Bower
<input type="checkbox"/>	CocoaPods-default	CocoaPo...
<input type="checkbox"/>	Debian-default	Debian
<input type="checkbox"/>	Docker-default	Docker
<input type="checkbox"/>	Gradle-default	Gradle
<input type="checkbox"/>	Ivy-default	Ivy
<input type="checkbox"/>	Maven-default	Maven
<input type="checkbox"/>	Npm-default	npm
<input type="checkbox"/>	NuGet-default	NuGet
<input type="checkbox"/>	Opkg-default	Opkg
<input type="checkbox"/>	Yum-default	YUM
<input type="checkbox"/>	Sbt-default	SBT
<input type="checkbox"/>	Vagrant-default	Vagrant

Cancel < Back Next > **Save & Finish**

Filtering and deleting

Start typing the name of a rule in the filter box to find the rule you are looking for.

Hover over a rule and click the delete icon on the right to delete it, or select a number of rules in the left column and click **Delete** to remove several rules at a time.

Rule Order

The order in which rules are displayed specifies the order in which they are applied. To change the rule order, you can select a rule and drag it to a new location in the list.

Conflicting rules

While it is possible to specify rules that conflict, since rules are applied in the order in which they appear, the first rule that is applied will take precedence.

Specifying Rule Parameters

To specify rule parameters, click **New** for a new rule, or the **Name** of a rule you want to edit.

Edit Rule ✕

Name *

Artifactory Input

Package Type *

yum YUM ▾

Available Tokens

- `\${architecture}`
- `\${artifactPath}`
- `\${packageName}`
- `\${packageVersion}`

Repository Filter

Path Filter

Bintray Output

Repository *

Package *

Close
Save

Name	A logical name for this rule
Artifactory Input	Parameters that determine which packages in Artifactory this rule applies to.
Package Type	Specifies the artifact package type for which this rule should be applied. Artifacts of other package types are ignored
Available Tokens	According to the package type selected, this specifies which tokens can be used to specify the deployment path in Bintray
Repository Filter	Wildcard expression that specifies for which original source repositories this rule should be applied. Packages in other repositories are ignored.
Path Filter	Wildcard expression that specifies the artifact path for which this rule should be applied. Packages that have a different path are ignored.
Output Bintray	Parameters that determine how artifacts should be deployed to Bintray.
Repository	The Bintray repository to which artifacts should be deployed.

Package	The Bintray package in the specified Bintray repository to which the artifact should be deployed. If the package does not exist, it will be created
Version	The Bintray version in the specified Bintray package and repository to which the artifact should be deployed. If the version does not exist, it will be created.
Path	The path in Bintray into which the artifact should be deployed in the specified repository.

Repository and Path Filter Parameters

Rules give you enormous flexibility in how you deploy artifacts to Bintray through use of regular expressions with capture groups where the capture groups may be named or unnamed.

Unnamed capture groups are back-referenced using placeholder tokens, while named capture groups are back-referenced using their names.

Using Unnamed Capture Groups

For each regular expression used in **Repository Filter** or **Path Filter** fields of the **Artifact Input** section, you can place tokens in fields of the **Bintray Output** section to back-reference them.

Tokens are written using the following format:

```

${source:x}
```

where:

source	path: A wildcard from the Path Filter field should be replaced repo: A wildcard from the Repository Filter field should be replaced
x	The wildcard number.

For example,

`${path:1}` means replace the first regular expression in the **Path Filter** field

`${repo:2}` means replace the second regular expression in the **Repository Filter** field

Example 1

Under Artifact Input, set **Path Filter** = `jfrog-(.*)rpm`

Under Bintray Output, set **Repository** = `rpm-${path:1}`

With these settings, a package called `jfrog-artifactory.rpm` will be deployed to a repository in Bintray called `rpm-artifactory`, while a package called `jfrog-mission-control.rpm` will be deployed to a repository in Bintray called `rpm-mission-control`

Example 2

Under Artifact Input, **Repository Filter** = `libs-(.*)`

Under Bintray Output, **Repository** = `rpm-${repo:1}`

With these settings:

A package called `jfrog-artifactory.rpm` from `libs-release-local` will be deployed to a repository in Bintray called `rpm-release-local`.

A package called `jfrog-artifactory.rpm` from `libs-snapshot-local`, will be deployed to a repository in Bintray called `rpm-snapshot-local`.

Using Named Capture Groups

You can give capture groups in the **Repository Filter** or **Path Filter** fields of the **Artifact Input** section specific names and then back-reference the capture group using its name in the fields of the **Bintray Output** section.

Named capture groups are written using the following format:

```
(?<name>regex)
```

where:

name	A logical name for the capture group
regex	The regular expression that defines repositories or paths that should pass through the filter

Once capture groups are defined, you can back-reference them in the fields of the Bintray Output section using the following format:

```
$(source:name)
```

where:

source	path: A capture group from the Path Filter field should be back-referenced repo: A capture group from the Repository Filter field should be back-referenced
name	The name of the capture group.

Example 1

If you set:

Under Artifactory Input, **Repository Filter** = (?<myRepo>-local)

Under Bintray Output, **Repository**= generic-\$(repo:myRepo)

Then a package in repository *builds-local* will be deployed to a repository in Bintray called *generic-builds*.

Example 2

If you set:

Under Artifactory Input, **Path Filter** = jfrog-(?<myPath>.*).rpm

Under Bintray Output, **Repository**= rpm-\$(path:myPath)

Then a package called *jfrog-artifactory.rpm* in repository *libs-release-local* will be deployed to a repository in Bintray called *rpm-artifactory*, while a package called *jfrog-mission-control.rpm* will be deployed to a repository in Bintray called *rpm-mission-control*.

Enumerating Capture Groups

You can use multiple capture groups when specifying rule parameters, and they are enumerated in the order in which they are received .

For example, if you set:

Under Artifactory Input, **Path Filter** = jfrog-(.*).(.*)

Under Bintray Output, **Repository** = \${path:2}-\${path:1}

Then:

jfrog-artifactory.rpm will be deployed to a repository in Bintray called *rpm-artifactory*

jfrog-artifactory.zip will be deployed to a repository in Bintray called *zip-artifactory*

jfrog-mission-control.rpm will be deployed to a repository in Bintray called *rpm-mission-control*

jfrog-mission-control.zip will be deployed to a repository in Bintray called *zip-mission-control*

You can even mix both named and unnamed capture groups together when specifying rule parameters, however, they are still enumerated in the order in which they are received.

Expanding the example above, if you set:

Under Artifactory Input, **Path Filter** = jfrog-(?<type>.*).(.*)

Under Bintray Output, **Repository** = \${path:2}-\${path:type}

"type" is enumerated as the first capture group, and it is back-referenced using its name.

The unnamed capture group is unnamed and second in the enumeration order, so it is back-referenced using its number, 2.

Distributing Artifacts

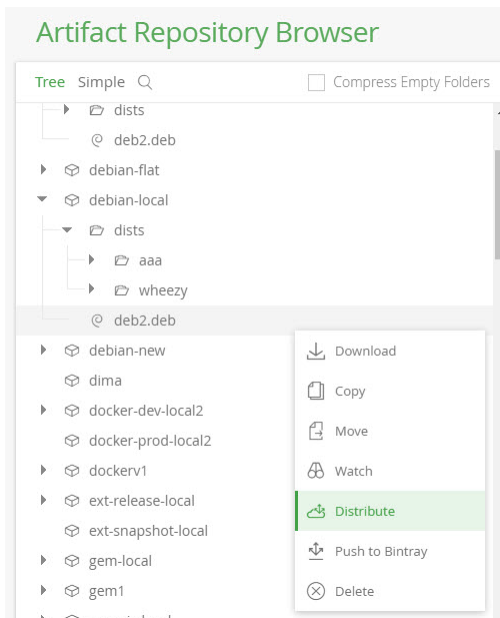
Once you have your distribution repositories configured, the last step in getting your files to Bintray is to invoke distribution. When you invoke distribution, Artifactory goes through the rules defined for your distribution repository in order until the artifact you are trying to distribute passes one of them, and then uploads the file to Bintray in accordance with that rule. Once distributed, the file will also appear in your distribution repository to indicate that it has been uploaded to Bintray.

There are two ways to do this:

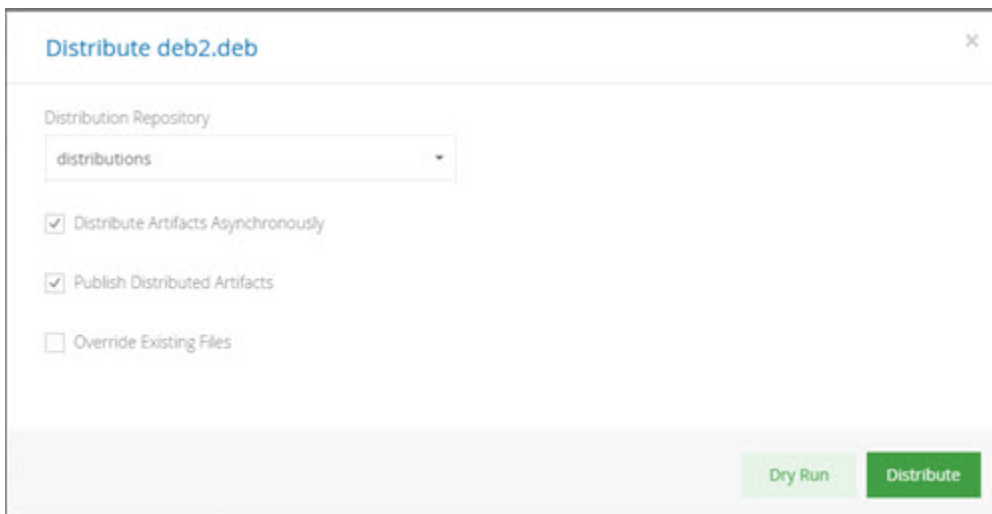
1. Distributing through the UI
2. Distributing via REST API

Distributing Through the UI

To distribute a file through the UI, select it in the Artifact Repository Browser and click **Distribute** in the right-click menu.



Artifactory will pop up the Distribution dialog where you can set final parameters for distribution.

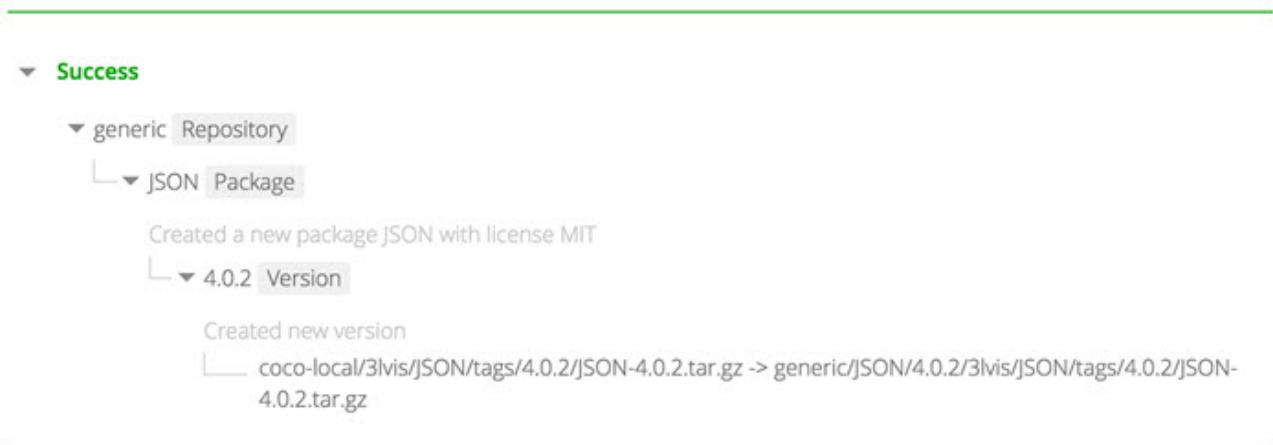


Distribution Repository	The distribution repository through which the artifact should be uploaded to Bintray. The rules defined for this repository will govern if/how the file is uploaded.
Distribute Artifacts Asynchronously	When checked, the file will be uploaded asynchronously. To verify upload to Bintray succeeded, refresh your distribution repository to see the distributed file. When unchecked, if upload to Bintray fails, Artifactory will display an error message. If upload to Bintray fails, please refer to the Artifactory System Log for details.
Publish Distributed Artifacts	When checked, files uploaded to Bintray are published to make them available for download by end users
Override Existing Files	When checked, the uploaded file will override another file with the same name if it exists in the upload path.

Unticking **Distribute Artifacts Asynchronously** will produce a UI screen representing the progress and summary of the distribution process:

Once distribution is complete, Artifactory displays the outcome of the process in a **Success** section and an **Error** section.

The **Success** section displays the repository, package and version in which the distributed artifacts will be hosted in Bintray. If the distribution process created a package or version, this will also be indicated in the Success section. In the example below, both a package and a version were created.



If an error occurs in moving distributed files to Bintray, the Distribution dialog will also display an **Errors** section. In the example below, the success section shows that there was an attempt to upload files to version `4.0.2` in package `JSON` in repository `generic`. The **Errors** section provides a detailed error message explaining the failure.



Errors

File `coco-local/3lvis/JSON/tags/4.0.2/JSON-4.0.2.tar.gz` resulted with the following errors

- Error distributing `coco-local/3lvis/JSON/tags/4.0.2/JSON-4.0.2.tar.gz` -> `generic/JSON/4.0.2/3lvis/JSON/tags/4.0.2/JSON-4.0.2.tar.gz: 409, Conflict Unable to upload files: An artifact with the path '3lvis/JSON/tags/4.0.2/JSON-4.0.2.tar.gz' already exists`

Dry Run

A dry run simulates the act of moving your selected files to Bintray without actually distributing them. This is a good way to ensure your configuration is correct and that there is no impediment to moving your files to Bintray.

Click "Dry Run" to start the simulation. Once completed, Artifactory will display the **Success** and **Error** sections as if the files were actually distributed.

Don't get confused

In a dry run, no files are moved; it's just a simulation

Distributing via REST API

JFrog Artifactory exposes a REST API that lets you automate deploying artifacts to Bintray. For details please refer to [Distribute Artifact](#) and [Distribute Build](#).

Deploying Snapshots to oss.jfrog.org

Overview

What is OJO

[oss.jfrog.org](#), or **OJO** for short, is an [Artifactory Cloud](#) instance for hosting your maven-compatible build snapshots, provided free of charge for selected opensource software projects.

All projects in OJO are public (i.e., all the artifacts and builds can be viewed by anyone).

Existing Bintray users are granted deploy permissions to relevant folders in Artifactory, according to the Maven Group ID of the packages they build.

Target Audience

This page is designed to help OSS contributors who want a free repository to host build snapshots, and eventually publish release versions to distribution via [Bintray](#).

At a Glance

The process of on-boarding to OJO, working with it to deploy continuous snapshots, and finally, promoting these snapshots from OJO to Bintray for distribution involves three simple steps:

1. [Creating an account on OJO](#)
2. [Building and deploying to the OJO Artifactory](#)
3. [Promoting a snapshot build to Bintray](#)

Page Contents

- Overview
 - What is OJO
 - Target Audience
 - At a Glance
- Getting Started with OJO
- Working with OJO
 - Resolving from and Publishing to OJO
 - Releasing to Bintray
 - Promoting a Release Build

Getting Started with OJO

To get account on OJO you must first have an account on Bintray.

1. Create a Maven repo on Bintray if it does not exist yet, and create your package inside this repo.
You can give your package any logical name, for example: `maven2gradle`
2. Ask for inclusion of the package in JCenter, by clicking the "**Add to JCenter**" button in the package main page.
In the request form, check "**Host my snapshot build artifacts on the OSS Artifactory at <https://oss.jfrog.org>**" and enter the desired Maven group ID for your package.
For example: `org.github.jbaruch.maven2gradle`

Request to include the package 'maven2gradle' in 'jcenter'

Host my snapshot build artifacts on the OSS Artifactory at <https://oss.jfrog.org>

This allows you to have your project's builds snapshots deployed to <https://oss.jfrog.org> and to release and publish them to Bintray in one click.

Enter a Maven group ID under which your artifacts can be uploaded. Your groupId is expected to be uniquely used by you. For example: 'org.acme.space-utils'.

`org.github.jb.maven2gradle`

Comments

Send

Feedback

After your request has been approved by the Bintray Team (usually within a few hours), you'll receive a confirmation email on the inclusion of your package in JCenter and the creation of your new OJO account.

OJO is Artifactory!

OJO is just a regular Artifactory Pro server, so [getting familiar with Artifactory](#) is recommended.

Bintray Organizations Support

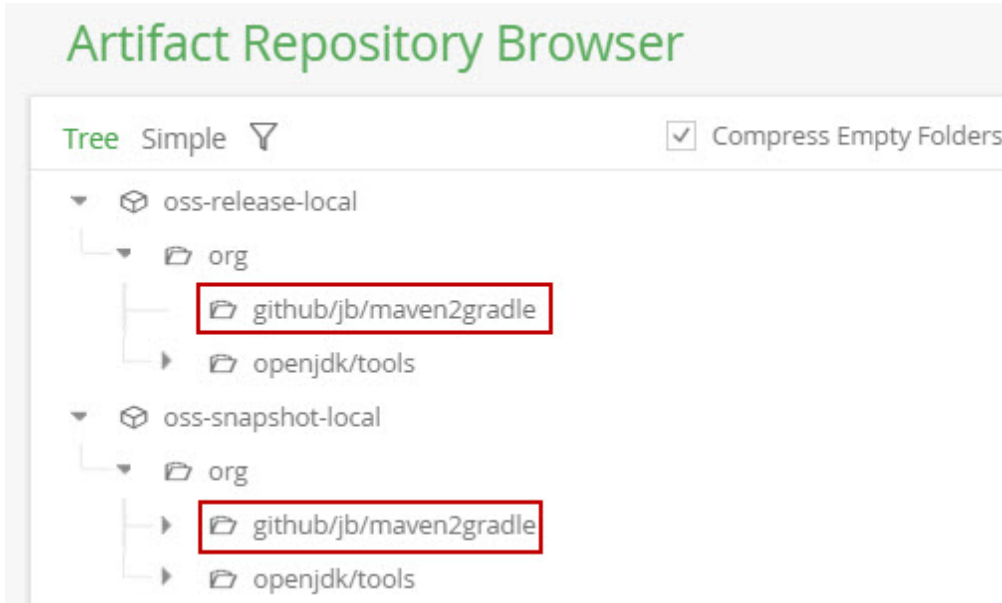
When requesting an OJO account for a repository belonging to an Bintray organization, the permissions in OJO will be granted to all the organization members, not only the member who asked for the OJO account.

Working with OJO

Once your OJO account has been created, you (and all the team members in the case of an organization) should be able to login to OJO using

your **Bintray username** and **API key** as the password.

You will see that a folder corresponding to the Maven Group ID has been created in OJO in the `oss-release-local` and the `oss-snapshot-local` repositories:



You have deploy permissions to these folders:

User Permissions

Filter by Permission Target

< page 1 of 1 >

Permission Target	Applied To	Repositories	Manage	Delete/Overwrite	Deploy/Cache	Annotate	Read
Anything	jb	1 ANY					✓
jb	jb	1 ANY		✓	✓	✓	✓

Resolving from and Publishing to OJO

There is nothing unusual about working with repositories in OJO. You can configure your build tool to resolve release and snapshot dependencies from the `libs-release` and the `libs-snapshot` OJO virtual repositories, respectively; and to deploy build snapshots to the `oss-snapshot-local` repository. As long as the `<groupId>` in your pom (for Maven) or the `project.group` (for Gradle) matches the group ID you requested during onboarding, the deployment should succeed.

Please consult the Artifactory documentation on how to set up [Maven](#) or [Gradle](#) for resolution and deployment.

Artifactory Build Info is a must!

In order to release and promote snapshots to Bintray you need to deploy a Build Info BOM to Artifactory. The easiest way to achieve this automatically it is to use the [Build Integration](#) feature of Artifactory or the [Gradle Artifactory Plugin](#); These and other options are described in the next section.

Releasing to Bintray

Currently, you have two ways to deploy artifacts to Bintray:

1. Promoting a Release Build

This will promote snapshot artifacts to release and then deploy them to Bintray:

- [Use promotion from the Jenkins Artifactory plugin](#) - This allows you to use the Jenkins UI to promote the snapshot artifacts from a selected job run.
- [Invoking promotion with REST](#) - This allows promotion of a build created with any build server/tool and full programatic automation of the promotion process.

2. Uploading Release Artifacts

Directly upload deployed release artifact to Bintray. If you have a released version of an artifact or a build, you can deploy them to Bintray using the regular [Bintray integration](#).

Promoting a Release Build

Promoting a Build from Jenkins

Promotion from Jenkins is performed by invoking a custom "**snapshotsToBintray**" promotion plugin. Here's what you need to do:

1. Install the [Jenkins Artifactory Plugin](#) and configure Artifactory servers and repositories as described in the [Jenkins documentation](#). You should configure the `oss-release-local` and `oss-snapshot-local` as release and snapshot targets, respectively.
2. In your build configuration, add the "Deploy artifacts to Artifactory" post-build action and check "Deploy Maven artifacts", "Capture and publish build info" and "Allow promotion of non-staged builds":

Post-build Actions

Deploy artifacts to Artifactory

Artifactory server

Target releases repository

Target snapshots repository

Custom staging configuration

Override default deployer credentials

Deploy even if the build is unstable

Deploy maven artifacts

Include Patterns

Exclude Patterns

Deployment properties

Capture and publish build info

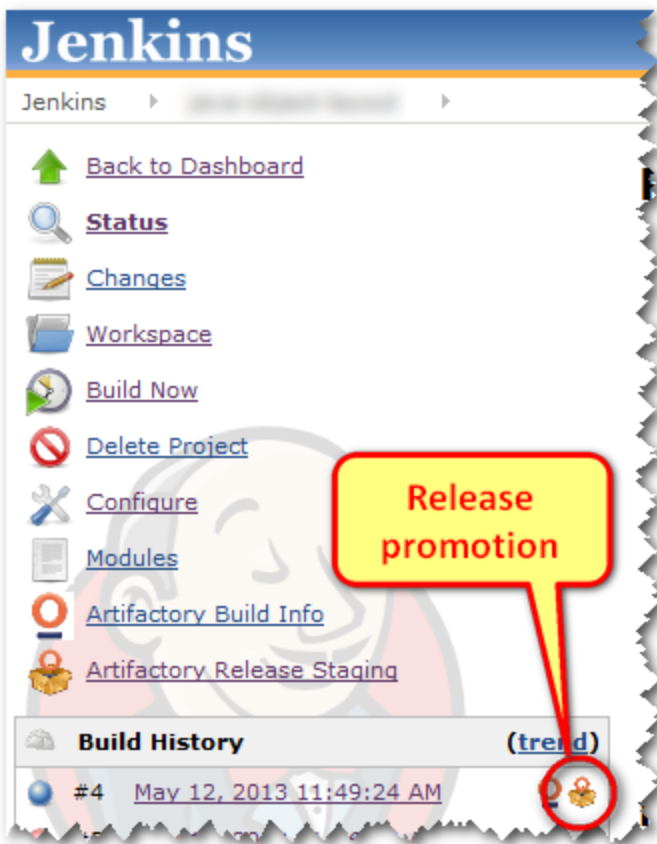
Include environment variables

Include Patterns

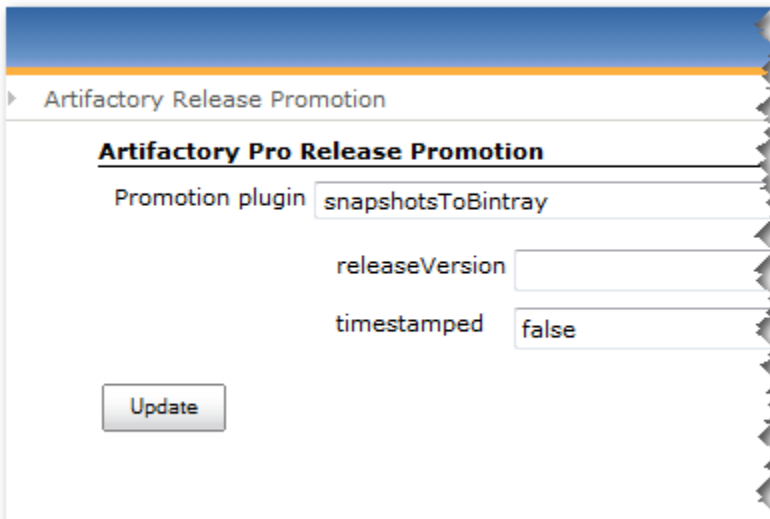
Exclude Patterns

Allow promotion of non-staged builds

3. Run your build. Upon successful completion, the build result page will have a link to the "Artifactory Release Promotion" action:



4. In the promotion configuration screen, select "snapshotsToBintray" promotion plugin:



There are two parameters to configure here:

- a. Override the release version. By default, the version is calculated by removing the -SNAPSHOT suffix from the snapshot version, e.g. 1.0-SNAPSHOT will be released to Bintray as version 1.0. Specifying a value in this field overrides the default version scheme.
- b. Append a timestamp to the version. This will add a timestamp string (in Maven's timestamp format: yyyyMMdd.HH:mm:ss) to the release version. Values of true, y or 1 will cause the timestamp suffix to be appended.

5. Click the "Update" button. Your release artifacts are now uploaded to Bintray.

Promoting a Build Using REST API

If you don't use Jenkins or if you need fully automated promotion, you can issue an HTTP PUT request that will trigger promotion and release to Bintray. Promotion still operates on a Build Info BOM, previously saved in Artifactory. Here's what you need to do:

1. Deploy a build to Artifactory in one of the following ways:
 - a. Using a build server with an Artifactory plugin. Plugins currently exist for Jenkins, Hudson, Bamboo and TeamCity. Please see the [Artifactory Build Integration documentation](#) for further instructions on getting the build info BOM into Artifactory.
 - b. Using the [Gradle Artifactory Plugin](#).
 - c. Configure Maven to use Artifactory Listener as described [here](#).
2. Execute the [build promotion plugin](#) call. The call accepts the same parameters as the [invocation of Jenkins promotion plugin](#). Here's an example using CURL:

```
curl -X POST -u bintrayUser:apiKey  
http://oss.jfrog.org/api/plugins/build/promote/snapshotsToBintray/buildName/3
```

Log Analytics

Overview

The Sumo Logic App for Artifactory automatically creates an account with Sumo Logic and sends it logs for analysis. Once an account is created through Artifactory, you can connect several instances to that same account. The Sumo Logic data analytics platform will, in turn, display pre-enabled dashboards which can be customized as needed. This enables you to access Sumo Logic's premium operational analytics directly from Artifactory, letting you index and analyze both structured metrics data and unstructured log data together in real time. The Sumo Logic App for Artifactory provides you with a customizable dashboard showing a variety of analytics such as:

- Traffic by geo-location
- Active IPs
- Most active repositories
- Top referred files
- Requests by status codes
- Denied login attempts
- and much more.

For more details about the different analytics that the Sumo Logic App for Artifactory can provide, please visit the [Sumo Logic website](#).

Page Contents

- [Overview](#)
- [Configuration](#)
 - [Credentials](#)
 - [Creating a New Account](#)
 - [Using an Existing Account](#)
 - [Connection Established](#)
 - [Traffic Log](#)
- [Webinar](#)

Configuration

The Log Analytics Configuration can be found in the **Admin** module under **Advanced | Log Analytics**.

Log Analytics Configuration

Sumo Logic Integration

The JFrog Artifactory / Sumo Logic integration gives you a centralized overview of your artifact repositories with the ability to drill down and quickly identify recent changes, check application dependencies and identify potential issues. Through dashboards, queries and searches that are pre-enabled out-of-the-box, Sumo Logic allows you to analyze all data that Artifactory generates. For a complete overview, [click here](#).



Artifactory Dashboard Settings

Enable

Create New Connection
Create an Artifactory dashboard

Use Existing Client ID and Secret
Connect this instance to an existing Artifactory dashboard

Client ID *



Secret *



Proxy

Access Dashboard

Get started by setting the **Enable** checkbox.

Continue by obtaining your Sumo Logic credentials as described in the next section.

Credentials

To access the Sumo Logic Artifactory dashboard, you need a Client ID and Secret.

Creating a New Account

If this is the first time you are using the Sumo Logic App for Artifactory you need to create an account to obtain your login credentials. Select **Create New Connection**, and click "Access Dashboard".

Artifactory will connect you to the Sumo Logic App for Artifactory. Your user name will automatically be taken as the email from your Artifactory account.

The image shows a Sumo Logic login form. At the top left is the Sumo Logic logo, which consists of a blue square with a white plus sign followed by the text "sumologic" in a blue sans-serif font. Below the logo is the text "Get free access to your dashboards". There is a text input field containing the email address "@jfrog.com". Below the email field is a dropdown menu with "North America" selected. Underneath the dropdown is a checkbox that is checked, with the text "I agree to the Service License Agreement". At the bottom right of the form is a blue button with the text "Access Dashboard".

Select your region, agree to the **Service License Agreement** and click "Access Dashboard".

Region

For best performance, make sure to select the region that is geographically closest to your Artifactory instance.

Trial account is free for 30 days

The account created through this integration with Artifactory is a 30-day free trial account. To continue using the integration through this account beyond 30 days, you need to access your account on Sumo Logic and upgrade it to a paid account.

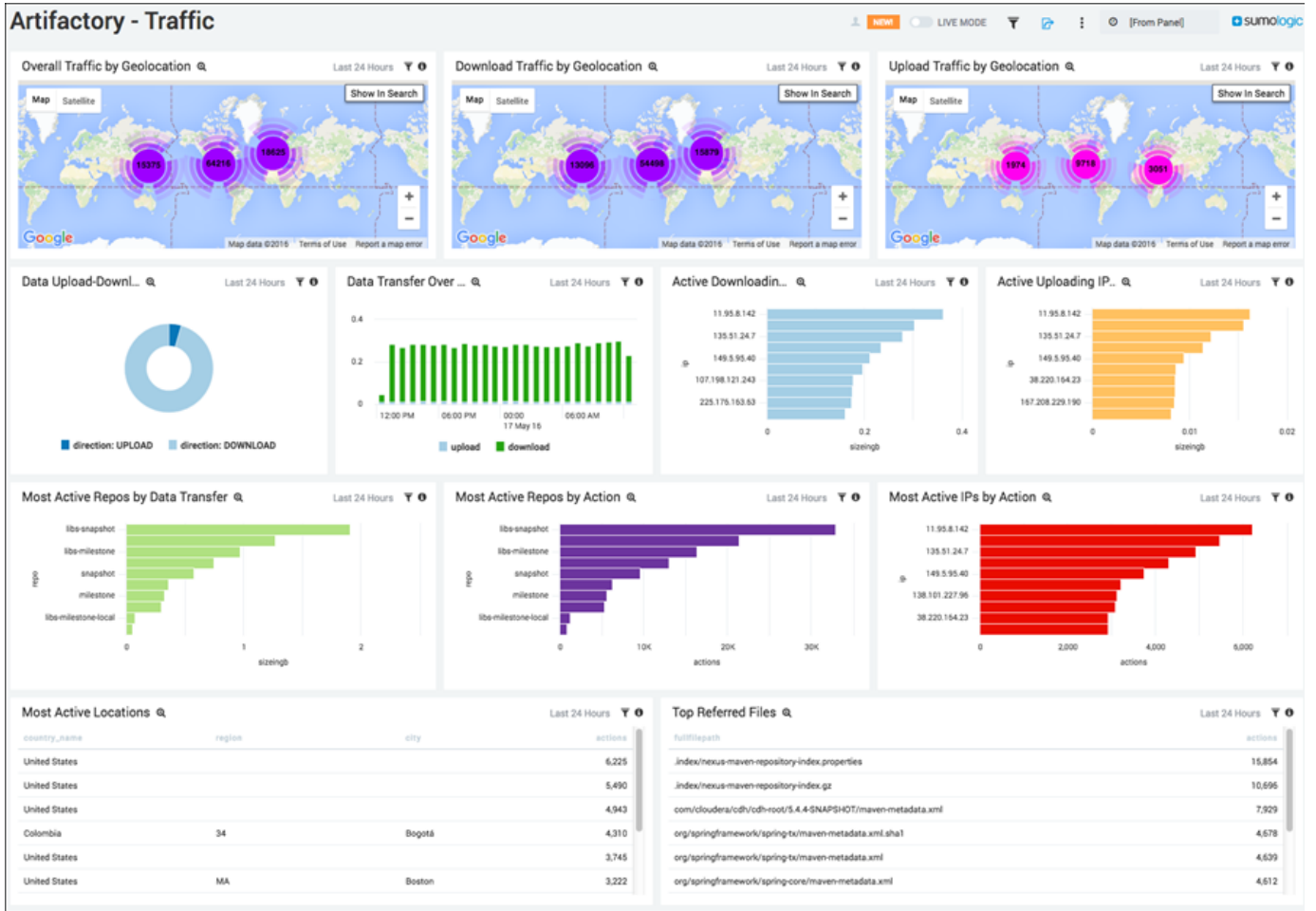
Using an Existing Account

If you already have a Sumo Logic account, you can use the credentials you already have and enter them in the corresponding fields (if you obtained them using the same browser you are currently using, they should be filled in automatically). Note that you may use credentials generated for the current Artifactory instance or from any other instance from which you have connected to Sumo Logic.

Connection Established

Once the connection is established, the Sumo Logic App will start analyzing your log files and populating the dashboard. This process may take several minutes to complete.

Out-of-the-box, the Sumo Logic App comes pre-configured with several dashboards. You can modify these and add to them as needed.



For details on how to work with and modify your dashboard, please refer to the Sumo Logic documentation.

The Log Analytics screen in Artifactory will reflect the connected status showing the Client ID and Secret needed to log into your account.

Log Analytics Configuration

Sumo Logic Integration

The JFrog Artifactory / Sumo Logic integration gives you a centralized overview of your artifact repositories with the ability to drill down and quickly identify recent changes, check application dependencies and identify potential issues. Through dashboards, queries and searches that are pre-enabled out-of-the-box, Sumo Logic allows you to analyze all data that Artifactory generates. For a complete overview, [click here](#).



Artifactory Dashboard Settings

Enable

Create New Connection

Create an Artifactory dashboard

Use Existing Client ID and Secret

Connect this instance to an existing Artifactory dashboard

Client ID *

cf0 [redacted] 7c



Secret *

32 [redacted] 74



Proxy

[Empty dropdown menu]

[Access Dashboard](#)

Traffic Log

To activate the traffic.log file, add the following parameter to your `ARTIFACTORY_HOME/etc/artifactory.system.properties` file :

```
artifactory.traffic.collectionActive=true
```

For this change to take effect so you can start collecting site traffic information, you will need to restart your system.

Webinar

For more details on how to set up and use Artifactory's integration with Sumo Logic, please watch the webinar below.

Artifactory Pro

Overview

Artifactory Pro exposes an extensive set of capabilities on top of the core repository management features that are available to you from Artifactory Open Source:

Pro and Enterprise Features

A wide range of features to support the needs for enterprise artifact management including security, high availability, replication, advanced search and more

Package Management	Full support for all major package formats and dependency managers
Ecosystem Integration	Integration with the build ecosystem and additional JFrog products
CI Server Integration	Integration with all major CI servers

Comparing ArtifactoryPro , Artifactory OSS and Artifactory Online

To compare the features and services offered by each version of Artifactory please refer to the [Artifactory Version Comparison Matrix](#) to see which version of Artifactory best fits your needs.

For more information please contact support@jfrog.com.

Page Contents

- [Overview](#)
 - [Package Management](#)
- [Download](#)
- [Installation and Upgrade](#)
- [Activating Artifactory Pro](#)

Read More

- [Artifactory Comparison Matrix](#)
- [Pro Features](#)
- [Package Management](#)
- [Ecosystem Integration](#)
- [Build Integration](#)

Download

If you need a license, please visit the JFrog website and either [purchase a license](#) or [request an evaluation license](#).

Once you submit the corresponding form, a download link will be provided to you by email.

You may also access the latest version through the [Artifactory Pro Download Site](#).

Installation and Upgrade

Performing a clean installation of Artifactory Pro is identical to installing Artifactory OSS. Please refer to [Installing Artifactory](#).

To upgrade from a previous version of Artifactory Pro or Artifactory OSS, please refer to [Upgrading Artifactory](#).

Data is preserved when upgrading from Artifactory OSS to Artifactory Pro

For a standalone installation, to upgrade an instance of Artifactory OSS to Artifactory Pro **of the same version** you only need to replace the `artifactory.war` file and enter a valid license key. All data stored in Artifactory is preserved in the process.

Once you have entered a valid Artifactory Pro license key, all Artifactory Pro features will be available with the same settings and content you had on the Artifactory OSS version from which you upgraded.

Activating Artifactory Pro

Whether you have requested an evaluation of Artifactory Pro, or have purchased a license, your license key is provided in the same email that contains the download link sent to you.

Your Artifactory administrator should enter the license key into the corresponding field in the **Admin** module under **Configuration | Register License**.

Administrator

You must be an Artifactory administrator in order to access the License Key field.

Artifactory License

License Details

Licensed to: JFrog
Valid Through: May 15, 2018
License Type: Commercial

License Key

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA...
-----END RSA PRIVATE KEY-----
```

Reset

Save

Using encrypted passwords

If you are using encrypted passwords with an IBM JDK/JRE, you may encounter encryption restrictions. For details please refer to [Using Your Secure Password](#).

Artifactory Comparison Matrix

Choose the Artifactory Edition that Fits You Best

	OSS	Pro	SaaS	SaaS (Dedicated Server)	Enterprise
Basic Artifact Management: Details... <ul style="list-style-type: none">Proxy and cache remote repository artifactsBulk artifact deployment (from archive)Include/exclude patterns for stored artifactsDeploy Artifacts via the UI or via REST/HTTPMove/copy/delete artifacts through the UIChecksum-based Storage with Deduplication	✓	✓	✓	✓	✓

On Demand Jar Signing and Web Start Application Hosting Custom repository layout for non-Maven module management Repository Replication					
Multi-push Replication					
Universal support for all major package formats: Maven Other package formats... <ul style="list-style-type: none"> • Bower • Chef Cookbooks • CocoaPods • Conan • Debian • Docker • Git LFS • NPM • NuGet • Opkg • P2 • PHP Composer • Puppet • PyPI • RPM • RubyGems • SBT • Vagrant • VCS 					
Integration with all leading CI-servers					
Promotion, demotion and cleanup of build artifacts Managing build artifacts for reproducible builds					
Powerful REST API for Release Automation Extend Artifactory with Groovy-based User Plugins					
Basic Security Details... LDAP Authentication Role-based authorization with teams and permissions					
LDAP Groups					
Multiple additional options for authentication Details... Active Directory, Atlassian Crowd and JIRA, OAuth (multiple providers) Automatic 3rd Party License Violation Detection per Build					
Powerful SSO integration for NTLM, Kerberos, Etc.					
Search by Name, Archive, Property or Checksum Values Artifactory Query Language (AQL)					
Annotate Artifacts with Searchable Properties Aggregate and Run Bulk Operations on Search Results					

Advanced Storage Solutions Details... S3 Object Storage Google Cloud Storage Microsoft Azure Cloud Storage Filestore Sharding			 (Managed by JFrog)	 (Managed by JFrog)	
High Availability Details... Five-nines Availability Redundant Cluster of Servers Unlimited Server Scalability Near-zero Maintenance Downtime			 (Managed by JFrog)	 (Managed by JFrog)	
Integration with Other JFrog Products JFrog Bintray and JFrog CLI JFrog Xray JFrog Mission Control	 (view only)	 (view only)	 (view only)	 	
Disaster Recovery					
SaaS Features Details... SaaS-based Maintenance-free Hosted Repository Always up-to-date Artifactory Version Setup Free Automated Backups					
Maintenance and Administration Details... Incremental and Historical Backup Services Focused Email Notifications for Artifact Changes Free Upgrades					
SLA-based Support		 (Pro Plus and Pro X)			

Pro Features

Overview

Artifactory Pro exposes a full set of capabilities that takes you beyond basic repository management to include advanced features for security, build integration, replication, advanced search, automation through a REST API and more.

Artifactory Query Language	A simple way to formulate complex queries that can find artifacts based on any number of search criteria.
Black Duck Code Center integration	Automate security and license governance of open source components and software.
Filtered Resources	Provision common settings and configuration to clients by turning any textual artifact into a dynamic template based on request parameters, current user identity and artifact properties.
GPG Signing	Manage signing key pairs so you can sign packages in different formats for authentication.
JFrog CLI	A simple interface that automates access to Artifactory through a compact and smart client.
LDAP Groups	Synchronize your LDAP groups with Artifactory and leverage your existing organizational structure to manage group-based permissions.
License Control	Manage and control your organization's licensing policies for third-party dependencies used by your software.
Properties	Annotate your artifacts and folders with fully-searchable properties.
Repository Layouts	Define the layout by which software modules are identified in your repository for automatic cleanup of old versions and cross-repository layout conversion.
Repository Replication	Actively synchronize your repository content and metadata with remote Artifactory repositories using pull or push replication.
REST API	Automate your repository management and release life-cycle with a powerful REST API.
Smart Search	Save search results in a stash browser for easy access and perform bulk operations on the result set.
SSO	Integrate with SSO infrastructures such as Apache HTTPd , Atlassian Crowd and SAML .
User Plugins	Extend Artifactory by plugging in your own custom Groovy scripts.
Watches	Watch selected artifacts, folders, or repositories for any event, and receive email notifications on changes that are interesting to you.
Webstart and JAR Signing	Manage signing keys and sign JAR files for use with Java Web Start

Enterprise Features

When activated with an enterprise license, JFrog Artifactory Pro offers an additional set of features to meet the high-end needs for repository management in larger enterprises.

Filestore Sharding	Implement a sharded filestore for a flexible filestore that offers unmatched stability, unlimited scalability and optimized performance.
Google Cloud Storage	Let your Artifactory filestore reside with GCS for unlimited scalability, security and disaster recovery capabilities.
High Availability	Deploy Artifactory in a high availability configuration to maximize uptime (up to five-nines availability), manage heavy loads and minimize maintenance downtime.
Multi-push Replication	Replicate a repository to multiple remote sites simultaneously.
S3 Object Storage	Manage your filestore on the cloud with any S3 compliant provider such as Amazon S3.

Page Contents

- Overview
- Enterprise Features

Read more

- Artifactory Query Language
- Atlassian Crowd and JIRA Integration
- Azure Blob Storage
- Black Duck Code Center Integration
- Filestore Sharding
- Filtered Resources
- Google Cloud Storage
- GPG Signing
- LDAP Groups
- License Control
- OAuth Integration
- Properties
- Repository Layouts
- Repository Replication
- S3 Object Storage
- SAML SSO Integration
- Single Sign-on
- Smart Searches
- SSH Integration
- User Plugins
- Watches
- WebStart and Jar Signing

Artifactory Query Language

Overview

Artifactory Query Language (AQL) is specially designed to let you uncover any data related to the artifacts and builds stored within Artifactory. Its syntax offers a simple way to formulate complex queries that specify any number of search criteria, filters, sorting options, and output parameters. AQL is exposed as a RESTful API which uses data streaming to provide output data resulting in extremely fast response times and low memory consumption. Currently, AQL can only extract data that resides in your instance of Artifactory, so it runs on **local repositories, remote repository caches and virtual repositories**.

Here are a few simple examples:

```
// Return all artifacts of the "artifactory"
build.
items.find({"@build.name":{"$eq":"artifactory"}})

// Return all builds that have a dependency with
a license that is not Apache.
builds.find({"module.dependency.item.@license":{"
  $nmatch":"Apache-*"}})

// Return all archives containing a file called
"org/artifactory/Main.class".
items.find({"archive.entry.name":{"$eq":"Main.cla
ss"} ,
  "archive.entry.path":{"$eq":"org/artifactory"}})
```

Here is a slightly more complex example.

```
// Return all entries of any archive named
"Artifactory.jar" from any build named
"Artifactory" with
// build number 521.
archive.entries.find( {
  "archive.item.name": {"$eq": "Artifactory.jar"},

  "archive.item.artifact.module.build.name": {"$eq":
"Artifactory"},

  "archive.item.artifact.module.build.number": {"$eq":
": "521"}
})
```

Page Contents

- Overview
- Architecture
 - Supported Domains
- Usage
 - Syntax
 - Using Fields
 - Execution
- Entities and Fields
- Constructing Search Criteria
 - Field Criteria
 - Properties Criteria
 - Compounding Criteria
 - Matching Criteria on a Single Property (\$msp)
 - Comparison Operators
 - Using Wildcards
 - Using Wildcards with \$match and \$nmatch
 - "Catch all" Notation on Properties
 - Examples
 - Date and Time Format
 - Relative Time Operators
- Specifying Output Fields
 - Displaying All Fields
 - Displaying Specific Fields
 - Users Without Admin Privileges
 - Filtering Properties by Key
- Sorting
- Display Limits and Pagination
- Working With Virtual Repositories
 - Filtering on a Virtual Repository
 - Output Fields

Here is another example that shows the full power of AQL to mine information from your repositories in a way that no other tool can match.

```
// Compare the contents of artifacts in 2 "maven+example" builds
items.find(
{
  "name": {"$match": "multi2*.jar"},
  "$or": [
    {
      "$and": [
        {"artifact.module.build.name": {"$eq": "maven+example"}},
        {"artifact.module.build.number": {"$eq": "317"}}
      ]
    },
    {
      "$and": [
        {"artifact.module.build.name": {"$eq": "maven+example"}},
        {"artifact.module.build.number": {"$eq": "318"}}
      ]
    }
  ]
}).include("archive.entry")
```

▼ [Click to view the output of this query...](#)

```
{
  "results": [ {
    "repo": "ext-snapshot-local",
    "path": "org/jfrog/test/multi2/3.0.0-SNAPSHOT",
    "name": "multi2-3.0.0-20151012.205507-1.jar",
    "type": "file",
    "size": 1015,
    "created": "2015-10-12T22:55:23.022+02:00",
    "created_by": "admin",
    "modified": "2015-10-12T22:55:23.013+02:00",
    "modified_by": "admin",
    "updated": "2015-10-12T22:55:23.013+02:00",
    "archives": [ {
      "entries": [ {
        "entry.name": "App.class",
        "entry.path": "artifactory/test"
      }, {
        "entry.name": "MANIFEST.MF",
        "entry.path": "META-INF"
      } ]
    } ]
  }, {
    "repo": "ext-snapshot-local",
    "path": "org/jfrog/test/multi2/3.0.0-SNAPSHOT",
    "name": "multi2-3.0.0-20151013.074226-2.jar",
    "type": "file",
    "size": 1015,
    "created": "2015-10-13T09:42:39.389+02:00",
    "created_by": "admin",
    "modified": "2015-10-13T09:42:39.383+02:00",
    "modified_by": "admin",
    "updated": "2015-10-13T09:42:39.383+02:00",
    "archives": [ {
      "entries": [ {
        "entry.name": "App.class",
        "entry.path": "artifactory/test"
      }, {
        "entry.name": "MANIFEST.MF",
        "entry.path": "META-INF"
      } ]
    } ]
  } ]
}
```



```

    ]]
  }],
  "range" : {
    "start_pos" : 0,
    "end_pos" : 2,
    "total" : 2
  }
}
}

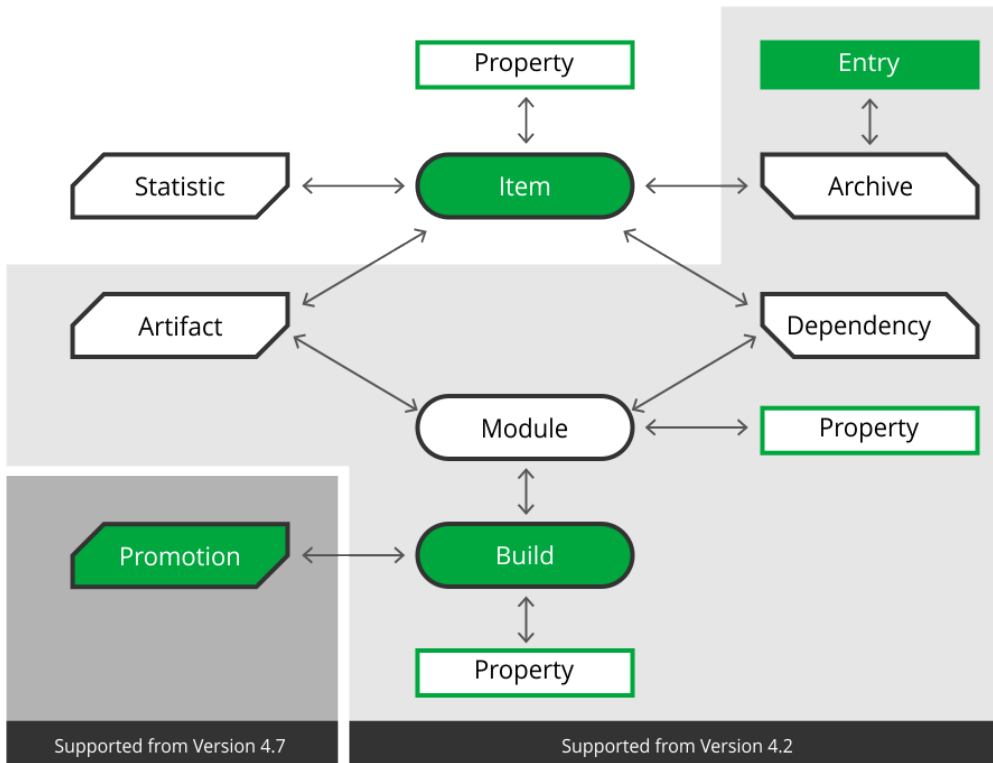
```

Architecture

AQL is constructed as a set of interconnected domains as displayed in the diagram below. You may run queries only on one domain at a time, and this is referred to as the **Primary** domain of the query.

Currently, the following are supported as primary domains: **Item**, **Build**, **Entry**, and **Promotion**. i.e., your queries may be of the form: **items.find(...)**, **builds.find(...)**, **archive.entries.find(...)**, or **build.promotions.find(...)**.

You may use fields from other domains as part of your search criteria or to specify fields to display in the output, but in that case, you need to follow the conventions described in [Using Fields](#).



Supported Domains

AQL was introduced in Artifactory V3.5.0 with support for **Item** as a primary domain with its attached **Property**, as well as **Statistic** as a secondary domain. Later versions of Artifactory introduced additional domains that can be included in queries. The following table summarizes from which version each domain is accessible.

	3.5.0	4.2.0	4.7.0
Item	✓	✓	✓
Item.Property	✓	✓	✓
Statistic	✓	✓	✓

<i>Archive</i>	✗	✓	✓
<i>Archive.Entry</i>	✗	✓	✓
<i>Artifact</i>	✗	✓	✓
<i>Dependency</i>	✗	✓	✓
<i>Module</i>	✗	✓	✓
<i>Module.Property</i>	✗	✓	✓
<i>Build</i>	✗	✓	✓
<i>Build.Property</i>	✗	✓	✓
<i>Promotion</i>	✗	✗	✓

Usage

Syntax

```
<domain_query>.find(<criteria>).include(<fields>).sort(<order_and_fields>).limit(<num_records>).offset(<offset_records>)
```

where:

<i>domain_query</i>	The query corresponding to the primary domain. Must be one of items , builds or entries .
<i>criteria</i>	The search criteria in valid JSON format
<i>fields</i>	(Optional) There is a default set of fields for query output. This parameter lets you specify a different set of fields that should be included in the output
<i>order_and_fields</i>	(Optional) The fields on which the output should be sorted, and the sort order. A default set of fields and sort order is defined for each domain.
<i>num_records</i>	(Optional) The maximum number of records that should be extracted. If omitted, all records answering the query criteria will be extracted.
<i>offset</i>	(Optional) The offset from the first record from which to display results (i.e. how many results should be skipped for display)

Limitation

Sort, **limit** and **offset** elements only work in the following cases:

- Your query does not have an **include** element
- If you do have an **include** element, you only specify fields from the primary domain in it.

For example, in the following query, **sort**, **limit** and **offset** will not work because the primary domain is **item**, but the **include** element specifies that fields from the **artifact**, **module** and **build** domains should be displayed:

```
items.find().include("artifact", "artifact.module", "artifact.module.build")
```

Using Fields

Any fields from your primary domain can be used directly anywhere in your query. If you use fields from other domains, they must be specified using a complete relation path from the primary domain.

For example, to find all items in a repository called "myrepo" you would use:

```
items.find({"repo": "myrepo"})
```

But to find all items created by modules named "mymodule" you would use:

```
items.find({"artifact.module.name" : "mymodule"})
```

And since you may also issue a query from the **build** domain, to find all builds that generated an item called "artifactory.war", you could also use:

```
builds.find({"module.artifact.item.name": "artifactory.war"})
```

Execution

To execute an AQL query, use the [Artifactory Query Language REST API](#).

Entities and Fields

You may issue a **find** request according to the [syntax](#) above, and configure your request to display fields from any of the domains.

Domain	Field Name	Type	Description
item	repo	String	The name of the repository in which this item is stored
	path	String	The full path associated with this item
	name	String	The name of the item
	created	Date	When the item was created
	modified	Date	File system timestamp indicating when the item was last modified
	updated	Date	When the item was last uploaded to a repository.
	created_by	String	The name of the item owner
	modified_by	String	The name of the last user that modified the item
	type	Enum	The item type (file/folder/any). If <code>type</code> is not specified in the query, the default type searched for is <code>file</code>

	depth	int	The depth of the item in the path from the root folder
	original_md5	String	The item's md5 hash code when it was originally uploaded
	actual_md5	String	The item's current md5 hash code
	original_sha1	String	The item's sha1 hash code when it was originally uploaded
	actual_sha1	String	The item's current sha1 hash code
	sha256	String	The item's sha256 hash code
	size	long	The item's size on disk
	virtual_repos	String	The virtual repositories which contain the repository in which this item is stored.
archive			The archive domain currently contains no fields
entry	name	String	The entry's name
	path	String	The path of the entry within the repository
promotion	created	Date	When the build was promoted
	created_by	String	The Artifactory user that promoted the build
	status	String	The status of the promotion
	repo	String	The name of the repository to which the build was promoted
	comment	String	A free text comment about the promotion
	user	String	The CI server user that promoted the build
build	url	String	The URL of the build
	name	String	The build name
	number	String	The build number
	created	Date	File system timestamp indicating when the item was last modified
	created_by	String	The name of the user who created the build
	modified	Date	File system timestamp indicating when the build was last modified
	modified_by	String	The name of the last user that modified the build
property	key	String	The property key
	value	String	The property value
stat	downloaded	date	The last time an item was downloaded
	downloads	int	The total number of downloads for an item
	downloaded_by	String	The name of the last user to download this item
	remote_downloads	int	The total number of downloads for an item from a smart remote repository proxying the local repository in which the item resides
	remote_downloaded	date	The last time an item was downloaded from a smart remote repository proxying the local repository in which the item resides
	remote_downloaded_by	String	The name of the last user to download this item from a smart remote repository proxying the local repository in which the item resides
	remote_origin	String	The address of the remote Artifactory instance along a smart remote proxy chain from which the download request originated.
	remote_path	String	The full path along a smart remote proxy chain through which the download request went from the origin instance to the current instance.

artifact	name	String	The name of the artifact
	type	String	The type of the artifact
	sha1	String	The SHA1 hash code of the artifact
	md5	String	The MD5 hash code of the artifact
module	name	String	The name of the module
dependency	name	String	The name of the dependency
	scope	String	The scope of the dependency
	type	String	The type of the dependency
	sha1	String	The SHA1 hash code of the dependency
	md5	String	The MD5 hash code of the dependency

Constructing Search Criteria

The **criteria** element must be a valid JSON format statement composed of the criteria that specify the items that should be returned. It is essentially a compound boolean statement, and only elements for which the statement evaluates to **true** are returned by the query.

Each criterion is essentially a comparison statement that is applied either to a field or a property. Please see the full list of [Comparison Operators](#). While each criterion may be expressed in complete general format, AQL defines shortened forms for readability as described below.

Field Criteria

The general way to specify a criterion on a field is as follows:

```
{"<field>" : {"<comparison operator>" : "<value>"}}
```

If the query applied is to a different domain, then field names must be pre-pended by a relation path to the primary domain.

For example:

```
//Find items whose "name" field matches the expression "*test.*"
items.find({"name": {"$match" : "*test.*"}})

//Find items that have been downloaded over 5 times.
//We need to include the "stat" specifier in "stat.downloads" since
downloads is a field of the stat domain and not of the item domain.
items.find({"stat.downloads":{"$gt":"5"}})

//Find items that have never been downloaded. Note that when specifying
zero downloads we use "null" instead of 0.
//We need to include the "stat" specifier in "stat.downloads" since
downloads is a field of the stat domain and not of the item domain.
items.find({"stat.downloads":{"$eq":null}})

//Find builds that use a dependency that is a snapshot
builds.find({"module.dependency.item.name":{"$match":"*SNAPSHOT*"}})
```

Fields with "Zero" value in the stat domain

Note that when searching for items that have a "zero" value in the stat domain, you should search for null, not 0. For example, as shown above, when searching for items with zero downloads you specify "null" instead of 0.

Short notation for Field criteria

AQL supports a short notation for search criteria on fields.

An "equals" ("=\$eq") criterion on a field may be specified as follows:

```
{ "<field>" : "<value>" }
```

Example	Find items whose "name" field equals "ant-1.9.4.jar"
Regular notation	<code>items.find({ "name": { "\$eq": "ant-1.9.4.jar" } })</code>
Short notation	<code>items.find({ "name": "ant-1.9.4.jar" })</code>

Properties Criteria

Artifactory lets you attach, and search on properties in three domains: **items**, **modules** and **builds**.

The general way to specify a criterion on a property is as follows:

```
{ "@<property_key>" : { "operator": "<property_value>" } }
```

Accessing the right properties

If you are specifying properties from the primary domain of your query, you may simply enter the property key and value as described above. If you are specifying properties from one of the other domains, you need to specify the full relational path to the property.

In the example below, the primary domain is the **build** domain, but we want to find builds based a property in the **item** domain, so we must specify the full path to the property:

```
builds.find({ "module.artifact.item.@qa_approved" : { "$ne" : "true" } })
```

Here are some examples:

```
//Find items that have been approved by QA"
items.find({ "@qa_approved" : { "$eq" : "true" } })

//Find builds that were run on a linux machine"
builds.find({ "@os" : { "$match" : "linux*" } })

//Find items that were created in a build that was run on a linux machine.
items.find({ "artifact.module.build.@os" : { "$match" : "linux*" } })
```

Short notation for properties criteria

AQL supports a short notation for search criteria on properties.

An "equals" ("=\$eq") criterion on a property may be specified as follows:

```
{ "@<property_key>" : "<property_value>" }
```

Example	Find items with associated properties named "license" with a value that equals "GPL"
Regular notation	<code>items.find({"@artifactory.licenses" : {"\$eq" : "GPL"}})</code>
Short notation	<code>items.find({"@artifactory.licenses" : "GPL"})</code>

Compounding Criteria

Search criteria on both fields and properties may be nested and compounded into logical expressions using "\$and" or "\$or" operators. If no operator is specified, the default is **\$and**

```
<criteria>={<"$and" | "$or">:[{<criteria>},{<criteria>}]}
```

Criteria may be nested to any degree

Note that since search criteria can be nested to any degree, you may construct logical search criteria with any degree of complexity required.

Here are some examples:

```

//This example shows both an implicit "$and" operator (since this is the
default, you don't have to expressly specify it, but rather separate the
criteria by a comma), and an explicit "$or" operator.
//Find all items that are files and are in either the jcenter or my-local
repositories.
items.find({"type" : "file", "$or": [{"repo" : "jcenter", "repo" : "my-local"
}]))

//Find all the items that are either in a repository called "debian" and
whose name ends with ".deb" or are in a repository called "yum" and whose
name ends with ".rpm".
items.find(
{
  "$or":
  [
    {
      "$and":
      [
        {"artifact.module.build.name" : "my_debian_build"} ,
        {"name" : {"$match" : "*.deb"}}
      ]
    },
    {
      "$and":
      [
        {"artifact.module.build.name" : "my_yum_build"} ,
        {"name" : {"$match" : "*.rpm"}}
      ]
    }
  ]
}
)

//Find all items in a repository called "my_local" that have a property
with a key called "license" and value that is any variant of "LGPL".
items.find({"repo" : "my_local"}, {"@artifactory.licenses" : {"$match" :
"*LGPL*"}})

```

Matching Criteria on a Single Property (\$msp)

A search that specifies several criteria on properties may sometimes yield unexpected results.

This is because items are frequently annotated with several properties, and as long as any criterion is true for any property, the item will be returned in a regular **find**.

But sometimes, we need to find items in which a single specific property answers several criteria. For this purpose we use the **\$msp (match on single property)** operator.

The fundamental difference between a regular **find** and using the **\$msp** operator is:

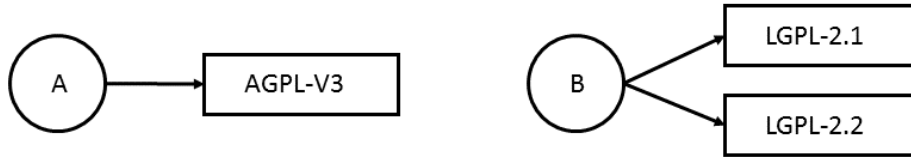
- **find** will return an item if **ANY** of its properties answer **ALL** of the criteria in the search term.
- **\$msp** will only return an item if at least **ONE** of its properties answers **ALL** of the criteria in the **\$msp** term.

Here is an example.

Consider two items A and B.

A has a license property with value **AGPL-V3**

B has two license properties . One is **LGPL-2.1**, and the other **LGPL-2.2**



Now let's assume we want to find items that use any variety of GPL license as long as it's NOT LGPL-2.1.

In our example we would expect to get both **Items A and B** returned since **A** has **AGPL-V3** and **B** has **LGPL-2.2**.

As a first thought, we might write our query as follows:

```
items.find({
  "@license": {"$match": "*GPL*"},
  "@license": {"$ismatch": "LGPL-2.1*"}
})
```

But this query only returns **item A**.

Item A is returned because it clearly answers both criteria: "@license":{"\$match": "*GPL*"} and "@license":{"\$ismatch": "LGPL-2.1*"}

Item B is not returned because it has the property license=LGPL-2.1 which does not meet the criterion of "@license":{"\$ismatch": "LGPL-2.1*"}

If we use the **\$msp** operator as follows:

```
"items.find({
  "$msp": [
    "@license":{"$match": "*GPL*"},
    "@license":{"$ismatch": "LGPL-2.1*"}
  ]}).
```

Then both **Item A and Item B** are returned.

Item A is returned because it has the @license property **AGPL-V3** which meets **both** the {"@license":{"\$match": "*GPL*"}} criterion and the "@license":{"\$ismatch": "LGPL-2.1*"}

Item B is returned because it has the @license property **LGPL-2.2** which also meets **both** the {"@license":{"\$match": "*GPL*"}} criterion and the "@license":{"\$ismatch": "LGPL-2.1*"}

Note that the `$msp` operator works equally well on all domains that have properties: **item**, **module** and **build**.

Comparison Operators

The following table lists the full set of comparison operators allowed:

Operator	Types	Meaning
\$ne	string, date, int, long	Not equal to
\$eq	string, date, int, long	Equals
\$gt	string, date, int, long	Greater than
\$gte	string, date, int, long	Greater than or equal to

\$lt	string, date, int, long	Less than
\$lte	string, date, int, long	Less than or equal to
\$match	string	Matches
\$nmatch	string	Does not match

For time-based operations, please also refer to [Relative Time Operators](#).

Using Wildcards

To enable search using non-specific criteria, AQL supports wildcards in common search functions.

Using Wildcards with \$match and \$nmatch

When using the "\$match" and "\$nmatch" operators, the "*" wildcard replaces any string and the "?" wildcard replaces a single character.

"Catch all" Notation on Properties

In addition to supporting "\$match" and "\$nmatch", AQL supports a notation that uses wildcards to match **any** key or **any** value on properties.

If you specify "@*" as the property key, then it means a match on any key.

If you specify "*" as the property value, then it means a match on any value

Example	Find items that have any property with a value of "GPL"
Regular notation	<code>items.find({"\$and" : [{"property.key" : {"\$eq" : "*"}}, {"property.value" : {"\$eq" : "GPL"}}]})</code>
Short notation	<code>items.find({"@*" : "GPL"})</code>

Example	Find any items annotated with any property whose key is "license" (i.e. find any items with a "license" property)
Regular notation	<code>items.find({"\$and" : [{"property.key" : {"\$eq" : "license"}}, {"property.value" : {"\$eq" : "*"}}]})</code>
Short notation	<code>items.find({"@artifactory.licenses" : "*"})</code>

Be careful not to misuse wildcards

Wildcard characters ("*" and "?") used in queries that do not conform to the above rules are interpreted as literals.

Examples

To avoid confusion, here are some examples that use the "*" and "?" characters explaining why they are interpreted as wildcards or literals.

Query	Wildcard or Literal	Explanation	What the query returns
<code>items.find({"name" : {"\$match" : "ant-1.9.4.*"}})</code>	Wildcard	Wildcards on fields are allowed with the \$match operator.	All items whose name matches the expression "ant-1.9.4.*"
<code>items.find({"name" : {"\$eq" : "ant-1.9.4.*"}})</code>	Literal	Wildcards on fields are only allowed with the \$match and \$nmatch operators.	Only find items whose name is literally "ant-1.9.4.*"
<code>items.find({"@artifactory.licenses" : "*"})</code>	Wildcard	For properties, this short notation is allowed and denotes any value	All items with a property whose key is "license"

<code>items.find({"@artifactory.licenses": "*GPL"})</code>	Literal	This is the short notation replacing the \$eq operator for properties, but it does not use the "catch all" notation for properties.	All items with a license whose value is literally <code>"*GPL"</code>
<code>items.find({"@artifactory.licenses": {"\$match": "*GPL*"}})</code>	Wildcard	Wildcards on properties are allowed with the \$match operator.	All items with a license matches the expression <code>"*GPL"</code>

Date and Time Format

AQL supports Date and Time formats according to a [W3C profile](#) of the ISO 8601 Standard for Date and Time Formats.

The complete date and time notation is specified as:

YYYY-MM-DDThh:mm:ss.sTZD (e.g., 2012-07-16T19:20:30.45+01:00)

Date/Time specified in partial precision is also supported: (i.e. specify just the year, or year and month, year, month and day etc.)

For example, the following query will return all items that were modified after July 16, 2012 at 30.45 seconds after 7:20pm at GMT+1 time zone:

```
//Find all the items that have been modified after
2012-07-16T19:20:30.45+01:00
items.find({"modified" : {"$gt" : "2012-07-16T19:20:30.45+01:00"}})

//Find all the builds that have were created after 2012-07-01
builds.find({"created" : {"$gt" : "2012-07-01"}})
```

For full details, please refer to the [W3C documentation](#).

Relative Time Operators

AQL supports specifying time intervals for queries using relative time. In other words, the time interval for the query will always be relative to the time that the query is run, so you don't have to change or formulate the time period, in some other way, each time the query is run. For example, you may want to run a query over the last day, or for the time period up to two weeks ago.

Relative time is specified using the following two operators:

<i>\$before</i>	The query is run over complete period up to specified time.
<i>\$last</i>	The query is run over period from the specified time until the query is run

Time periods are specified with a number and one of the following suffixes:

<i>milliseconds</i>	"mills", "ms"
<i>seconds</i>	"seconds", "s"
<i>minutes</i>	"minutes"
<i>days</i>	"days", "d"
<i>weeks</i>	"weeks", "w"
<i>months</i>	"months", "mo"

years	"years", "y"
--------------	--------------

For example, to specify five days, you could use "5d". To specify two weeks, you could use "2w".

Below are some examples using relative time operators:

```
//Find all the items that were modified during the last three days
items.find({"modified" : {"$last" : "3d"}})

//Find all the builds that were created up to two weeks ago (i.e. no later
than two weeks ago)
builds.find({"created" : {"$before" : "2w"}})
```

Specifying Output Fields

Each query displays a default set of fields in the result set, however you have complete control this and may specify which fields to display using an optional **include** element in your query.

You can even specify to display fields from other entities related to your result set.

Displaying All Fields

Use: `.include("")`

For example:

```
//Find all items, and display all the item fields
items.find().include("*")
```

Displaying Specific Fields

Each query displays a default set of fields in the output. Using the `.include` element you can override this default setting and specify any particular set of fields you want to receive in the output.

Use: `.include("<field1>", "<field2>"...)`

For example:

```
//Find all items, only display the "name" and "repo" fields
items.find().include("name", "repo")
```

You can also display specific fields from other entities associated with those returned by the query.

If you specify any field from the **item** domain, then this will override the default output setting, and only the **item** fields you expressly specified will be displayed.

If you only specify fields from the **property** or **stat** domains, then the output will display the default fields from the **item** domain, and in addition, the other fields you expressly specified from the **property** or **stat** domains.

For example:

```

//Find all items, and display the "name" and "repo" fields as well as the
number of "downloads" from the corresponding "stat" entity
items.find().include("name", "repo", "stat.downloads")

//Find all items, and display the default item fields fields as well as the
stat fields
items.find().include("stat")

//Find all items, and display the default item fields as well as the stat
and the property fields
items.find().include("stat", "property")

//Find all items, and display the "name" and "repo" fields as well as the
stat fields
items.find().include("name", "repo", "stat")

//Find all builds that generated items with an Apache license, and display
the build fields as well as the item "name" fields. Click below to view the
output of this query
builds.find({
    "module.artifact.item.@license": {"$match": "Apache*"}
})
).include("module.artifact.item.name")

```

Click to view the output of the last query

Note that the output displays the default fields of the "build" domain, and the "name" field from the item domain. Fields from the module and artifact domains are not displayed since they were not specified in the include element.

```

{
  "results": [ {
    "build.created": "2015-09-06T15:49:01.156+03:00",
    "build.created_by": "admin",
    "build.name": "maven+example",
    "build.number": "313",
    "build.url": "http://localhost:9595/jenkins/job/maven+example/313/",
    "modules": [ {
      "artifacts": [ {
        "items": [ {
          "name": "multi-3.0.0-20150906.124843-1.pom"
        } ]
      } ]
    } ]
  }, {
    "build.created": "2015-09-06T15:54:40.726+03:00",
    "build.created_by": "admin",
    "build.name": "maven+example",
    "build.number": "314",
    "build.url": "http://localhost:9595/jenkins/job/maven+example/314/",
    "modules": [ {
      "artifacts": [ {
        "items": [ {
          "name": "multi-3.0.0-20150906.124843-1.pom"
        } ]
      } ]
    } ]
  } ],
  "range": {
    "start_pos": 0,

```

```
    "end_pos" : 2,  
    "total" : 2  
  }  
}
```

Users Without Admin Privileges

To ensure that non-privileged users do not gain access to information without the right permissions, users without admin privileges have the following restrictions:

1. The primary domain in the query may only be **item**.
2. The following three fields must be included in the `include` directive: **name**, **repo**, and **path**.

Note, however, that once these restrictions are met, you may include any other accessible field from any domain in the `include` directive.

Filtering Properties by Key

As described above, the primary use of the `.include` element is to specify output fields to display in the result set.

This notion is applied in a similar way in regard to properties. Each item may be annotated with several (even many) properties. In many cases you may only be interested in a specific subset of the properties, and only want to display those.

So the **.include** element can be used to filter out unwanted properties from the result, and only display (i.e. "include") those you are interested in.

For example, to display all the properties annotating an item found :

```
//Find all items, and display the "name" and "repo" fields, as well as all  
properties associated with each item  
items.find().include("name", "repo", "property.*")
```

However, if you are only interested in a specific property (e.g. you just want to know the version of each item returned), you can filter out all other properties and only include the property with the key you are interested in:

```
//Find all items, and display the "name" and "repo" fields, as well as the  
key and value of the "version" property of each item  
items.find().include("name", "repo", "@version")
```

Sorting

AQL implements a default sort order, however, you can override the default and specify any other sort order using fields in your output by adding the `.sort` element to the end of your query as follows:

```
.sort({"<$asc | $desc>" : [{"<field1>", "<field2>",... ]})
```

You can only specify sorting on fields that are displayed in the output (whether they are those displayed by default or due to a `.include` element).

Here are some examples:

```
// Find all the jars in artifactory and sort them by repo and name
items.find({"name" : {"$match":"*.jar"}}).sort({"$asc" : ["repo","name"]})

// Find all the jars in artifactory and their properties, then sort them
by repo and name
items.find({"name" : {"$match":"*.jar"}}).include("@").sort({"$asc" :
["repo","name"]})
```

Display Limits and Pagination

Limitation

Note the important limitation on *sort*, *limit* and *offset* **described above**.

Using the *.limit* elements, you can limit the number of records that will be displayed by your query.

```
// Find all the jars in artifactory and sort them by repo and name, but
only display the first 100 results
items.find({"name" : {"$match":"*.jar"}}).sort({"$asc" :
["repo","name"]}).limit(100)
```

You can also implement pagination when you want to focus on a subset of your results using the *.offset* element.

```
//Run the same example, but this time, display up to 50 items but skipping
the first 100
items.find({"name" : {"$match":"*.jar"}}).sort({"$asc" :
["repo","name"]}).offset(100).limit(50)
```

Working With Virtual Repositories

From version 4.8.1, AQL supports virtual repositories. Since virtual repositories only contain items indirectly through the local repositories they include, several conventions have been laid down as described in the following sections.

Filtering on a Virtual Repository

You may limit queries to search in a specified virtual repository. In practice this means that the query will be applied to local repositories and remote repository caches included in the specified virtual repository.

For example, find all the items within any repository contained in a virtual repository called "my-virtual":

```
items.find({"repo" : "my-virtual"})
```

Output Fields

The **item** domain has a **virtual_repos** field which includes the virtual repositories in which a found item is contained. In general, to display this field, you need to expressly specify it in your query as an **output field**. However, if your query specifies a virtual repository as its search target, the **virtual_repos** field is implicitly included in the search results as an output field.

An item must be accessible in order to be found

A search query will only find an item in a virtual repository if it is accessible by that virtual repository. For example, the local repository that contains an item may specify and [include or exclude pattern](#) which prevents access to the item by the encapsulating virtual repository. In this case the search query will not find the item.

Atlassian Crowd and JIRA Integration

Overview

The integration between Artifactory and Crowd/JIRA allows you to delegate authentication requests to Atlassian Crowd/JIRA, use authenticated Crowd/JIRA users and have Artifactory participate in a transparent SSO environment managed by Crowd/JIRA.

Page Contents

- [Overview](#)
- [Usage](#)
- [Crowd Groups](#)

Usage

Crowd integration can then be configured in the **Admin** module under **Security | Crowd/JIRA**.

Crowd / JIRA Users Management Configuration

Server Settings

Enable Crowd / JIRA Users Management Integration

Users Management Server: **Crowd** | JIRA

Server URL *

http:// /crowd

Application Name *

artifactory

Application Password *

.....

Session Validation Interval (Min) * [?](#)

10

Use Default Proxy Configuration [?](#)

Auto Create Artifactory Users [?](#)

Test

Synchronize Groups

Search Group by Username (leave blank for *) [?](#)

0 Groups

Filter by Group Name

[Import](#) < Page 1 of 1 >

Group Name	사람 그룹	Description	Sync...

Field Name	Description
Enable Crowd / JIRA Users Management Integration	Set this checkbox to enable security integration with Atlassian Crowd or JIRA.
User Management Server	Select which User Management Server you are using.
Server URL	The full URL of the server to use.
Application Name	The application name configured for Artifactory in Crowd/JIRA.
Crowd Application Password	The application password configured for Artifactory in Crowd/JIRA.
Session Validation Interval	The time window, in minutes, in which the session does not need to be revalidated.
Use Default Proxy Configuration	If this checkbox is set and a default proxy definition exists, it is used to pass through to the Crowd/JIRA Server.

Auto Create Artifactory Users	<p>When automatic user creation is off, authenticated users will not be automatically created inside Artifactory. Instead, for every request from a Crowd/JIRA user, the user is temporarily associated with default groups (if such groups are defined), and the permissions for these groups applies.</p> <p>Without automatic user creation, you will need to manually create the user inside Artifactory in order to manage user permissions that are not attached to his default groups.</p>
Filter by Group Name	Filter the search to see only groups of the specified username. If unchecked, all Crowd groups are shown.

To enable Crowd/JIRA integration:

1. Select which User Management Server you are using. If you select JIRA, SSO will be disabled since it's not supported by JIRA.
2. Define Artifactory as a [Custom Application Client](#) inside Crowd.
3. Complete the Crowd server URL, and the application credentials defined in Step 1.
4. The session validation interval defines the principal token validity time in minutes. If left at the default of 0, the token expires only when the session expires.
5. If you are using JIRA User Server provide it's URL in the "Crowd Server URL" and check the "Use JIRA User Server". This will disable SSO, which is not supported by JIRA.
6. If you have a proxy server between the Artifactory server and the Crowd server, you may set the **Use Default Proxy Configuration** check-box.
7. You may instruct Artifactory to treat externally authenticated users as temporary users, so that Artifactory does not automatically create them in its security store. In this case, permissions for such users are based on the permissions given to auto-join groups.
8. Test the configured connection and save it.

System properties

Crowd configuration properties may be added to the run time system properties or to the `$ARTIFACTORY_HOME/etc/artifactory.system.properties` file.

NOTE that setting a configuration through properties overrides configurations set through the user interface.

Crowd Groups

To use Crowd/JIRA groups:

1. Set up a Crowd server for authentication as detailed above.
2. Verify your setup by clicking the **Refresh** button on the **Synchronize Crowd Groups** sub-panel. A list of available Crowd groups, according to your settings is displayed.
3. The groups table allows you to select which groups to import into Artifactory and displays the sync-state for each group. A group can either be completely new or may already exist in Artifactory.
4. Select and import the groups that you wish to import to Artifactory. Once a group is imported (synced) a new external Crowd group is created in Artifactory with the name of the group.

Synchronize Crowd Groups

Filter by: Username Group Name

import
< page 1 of 1 >

Group Name	Description	Synced

You can [Manage Permissions](#) on the synced Crowd groups in the same way you manage them for regular Artifactory groups.

Users association to these groups is external and controlled strictly by Crowd.

Ensure the Crowd group settings is enabled in order for your settings to become effective.

Azure Blob Storage

Overview

Requires an **Enterprise license**

From version 5.4, Artifactory supports managing your Artifactory filestore on the cloud with Azure Blob Storage providing you with:

1. Massive scalability

On the cloud, your Artifactory filestore is massively scalable. You may freely continue to upload files without having to install or maintain any file storage devices. You can even upload files larger than 5 GB using multi-part upload with the blob size limit currently at 4.75 TB. Currently, there is also a 500 TB limit on an Azure Blob Storage account.

2. Security

An Azure Blob Storage account offers a variety of security capabilities such as role-based access control, Azure Active Directory, in-transit security, Storage Service encryption and more. For full details on the security capabilities provided by Azure Blob Storage, please refer to [Azure Storage Security Overview](#).

3. Disaster recovery

Since your files are replicated and stored with redundancy, using Azure Blob Storage offers the capability for disaster recovery.

Support for Azure Blob Storage is included with **JFrog Enterprise Edition**.

In order to use Azure Blob Storage with Artifactory, make sure you first [install](#) or [upgrade](#) to Artifactory V5.4.0 or later.

Backup your system. Your current filestore will be deleted.

Setting up Artifactory to use Azure Blob Storage will delete all files in your current filestore.

If you already have a running installation of Artifactory, then before you setup Artifactory to use Azure Blob Storage and migrate your filestore to the cloud, we strongly recommend that you do a [complete system backup](#).

To learn more, please refer to [Introduction to Microsoft Azure Storage](#) in the Microsoft Azure documentation.

Page Contents

- Overview
- Setting up Artifactory to Use Azure Blob Storage
 - Setting Your License
 - Configuring Artifactory to Use Azure Blob Storage
 - Migrating Your Filestore

Setting up Artifactory to Use Azure Blob Storage

First time installation or upgrade

Whether you are [installing Artifactory](#) for the first time, or are moving your filestore to Azure Blob Storage in the context of [upgrading Artifactory](#), we recommend that you first do a standard installation of Artifactory using the default settings, or a standard upgrade using your current settings.

To move your Artifactory filestore to Azure Blob Storage, execute the following steps:

- Shut down Artifactory.
- Set your [enterprise license](#)
- [Configure Artifactory to use Azure Blob Storage](#)
- [Migrate your filestore to the cloud](#)
- Start up Artifactory

Setting Your License

To use Azure Blob Storage, your Artifactory installation needs to be activated with an [enterprise license](#).

Configuring Artifactory to Use Azure Blob Storage

To configure Artifactory to use Azure Blob Storage, you need to use the Azure Blob Storage binary provider described in [Configuring the Filestore](#).

Migrating Your Filestore

For an Artifactory HA cluster running version 5.0 and above, to migrate your filestore, please refer to [Migrating Data from NFS](#).

Black Duck Code Center Integration

Deprecated

As of Artifactory v5.0, integration of Black Duck Code Center has been deprecated.

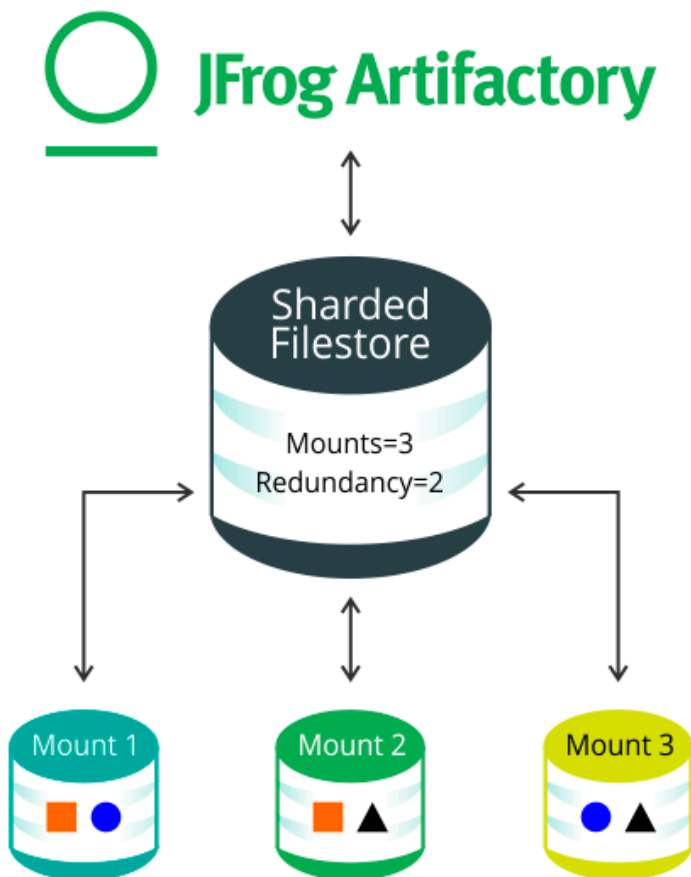
Access to Black Duck services is still available through its integration into JFrog Xray. For details, please refer to the [JFrog Xray documentation](#).

Filestore Sharding

Overview

From version 4.6, Artifactory offers a Sharding Binary Provider that lets you manage your binaries in a sharded filestore. A sharded filestore is one that is implemented on a number of physical mounts (M), which store binary objects with redundancy (R), where $R \leq M$.

For example, the diagram below represents a sharded filestore where $M=3$ and $R=2$. In other words, the filestore consists of 3 physical mounts which store each binary in two copies.



Artifactory's sharding binary provider presents several benefits:

Unmatched stability and reliability

Thanks to redundant storage of binaries, the system can withstand any mount going down as long as $M \geq R$.

Unlimited scalability

If the underlying storage available approaches depletion, you only need to add another mount; a process that requires no downtime of the filestore. Once the mount is up and running, the system regenerates the filestore redundancy according to configuration parameters you control.

Filestore performance optimization

Sharding Binary Provider offers several configuration parameters that allow you to optimize how binaries are read from or written to the filestore according to your specific system's requirements.

Enterprise license required

Sharded filestore is available for Artifactory installations activated with an enterprise license.

 Requires an **Enterprise license**

Page Contents

- [Overview](#)
- [Configuring a Sharding Binary Provider](#)
 - [Basic Sharding Configuration](#)
- [Using Balancing to Recover from Mount Failure](#)
- [Restoring Balance in Unbalanced Redundant Storage Units](#)
- [Optimizing System Storage](#)

Configuring a Sharding Binary Provider

A sharding binary provider is a binary provider as described in [Configuring the Filestore](#). [Basic sharding configuration](#) is used to configure a sharding binary provider for an instance of Artifactory Pro.

Basic Sharding Configuration

The following parameters are available for a basic sharding configuration:

<i>readBehavior</i>	This parameter dictates the strategy for reading binaries from the mounts that make up the sharded filestore. Possible values are: roundRobin (default): Binaries are read from each mount using a round robin strategy.
<i>writeBehavior</i>	This parameter dictates the strategy for writing binaries to the mounts that make up the sharded filestore. Possible values are: roundRobin (default): Binaries are written to each mount using a round robin strategy. freeSpace: Binaries are written to the mount with the greatest absolute volume of free space available. percentageFreeSpace: Binaries are written to the mount with the percentage of free space available.
<i>redundancy</i>	Default: r=1 The number of copies that should be stored for each binary in the filestore. Note that redundancy must be less than or equal to the number of mounts in your system for Artifactory to work with this configuration.
<i>concurrentStreamWaitTimeout</i>	Default: 30,000 ms To support the specified redundancy, accumulates the write stream in a buffer, and uses "r" threads (according to the specified redundancy) to write to each of the redundant copies of the binary being written. A binary can only be considered written once all redundant threads have completed their write operation. Since all threads are competing for the write stream buffer, each one will complete the write operation at a different time. This parameter specifies the amount of time (ms) that any thread will wait for all the others to complete their write operation. <div style="border: 1px solid green; padding: 5px; margin-top: 10px;">If a write operation fails, you can try increasing the value of this parameter.</div>

<p><i>concurrentStreamBufferKb</i></p>	<p>Default: 32 Kb The size of the write buffer used to accumulate the write stream before being replicated for writing to the “r” redundant copies of the binary.</p> <div style="border: 1px solid green; padding: 5px; margin-top: 10px;"> <p>If a write operation fails, you can try increasing the value of this parameter.</p> </div>
<p><i>maxBalancingRunTime</i></p>	<p>Default: 3,600,000 ms (1 hour) Once a failed mount has been restored, this parameter specifies how long each balancing session may run before it lapses until the next Garbage Collection has completed. For more details about balancing, please refer to Using Balancing to Recover from Mount Failure.</p> <div style="border: 1px solid green; padding: 5px; margin-top: 10px;"> <p>To restore your system to full redundancy more quickly after a mount failure, you may increase the value of this parameter. If you find this causes an unacceptable degradation of overall system performance, you can consider decreasing the value of this parameter, but this means that the overall time taken for Artifactory to restore full redundancy will be longer.</p> </div>
<p><i>freeSpaceSampleInterval</i></p>	<p>Default: 3,600,000 ms (1 hour) To implement its write behavior, Artifactory needs to periodically query the mounts in the sharded filestore to check for free space. Since this check may be a resource intensive operation, you may use this parameter to control the time interval between free space checks.</p> <div style="border: 1px solid green; padding: 5px; margin-top: 10px;"> <p>If you anticipate a period of intensive upload of large volumes of binaries, you can consider decreasing the value of this parameter in order to reduce the transient imbalance between mounts in your system.</p> </div>
<p><i>minSpareUploaderExecutor</i></p>	<p>Default: 2 Artifactory maintains a pool of threads to execute writes to each redundant unit of storage. Depending on the intensity of write activity, eventually, some of the threads may become idle and are then candidates for being killed. However, Artifactory does need to maintain some threads alive for when write activities begin again. This parameter specifies the minimum number of threads that should be kept alive to supply redundant storage units.</p>
<p><i>uploaderCleanupIdleTime</i></p>	<p>Default: 120,000 ms (2 min) The maximum period of time threads may remain idle before becoming candidates for being killed.</p>

Example 1

The code snippet below is a sample configuration for the following setup:

- A cached sharding binary provider with three mounts and redundancy of 2.
- Each mount "X" writes to a directory called **/filestoreX**.
- The read strategy for the provider is **roundRobin**.
- The write strategy for the provider is **percentageFreeSpace**.

```

<config version="4">
  <chain>
    <provider id="cache-fs" type="cache-fs">      <!-- This is a cached
filestore -->
    <provider id="sharding" type="sharding">      <!-- This is a
sharding provider -->
      <sub-provider id="shard1" type="state-aware"/> <!-- There
are three mounts -->
      <sub-provider id="shard2" type="state-aware"/>
      <sub-provider id="shard3" type="state-aware"/>
    </provider>
  </chain>

  // Specify the read and write strategy and redundancy for the sharding
binary provider
  <provider id="sharding" type="sharding">
    <readBehavior>roundRobin</readBehavior>
    <writeBehavior>percentageFreeSpace</writeBehavior>
    <redundancy>2</redundancy>
  </provider>

  //For each sub-provider (mount), specify the filestore location
  <provider id="shard1" type="state-aware">
    <fileStoreDir>filestore1</fileStoreDir>
  </provider>

  <provider id="shard2" type="state-aware">
    <fileStoreDir>filestore2</fileStoreDir>
  </provider>

  <provider id="shard3" type="state-aware">
    <fileStoreDir>filestore3</fileStoreDir>
  </provider>
</config>

```

Example 2

The following code snippet shows the "double-shards" template which can be used as is for your binary store configuration.

```

<config version="4">
  <chain template="double-shards" />

  <provider id="shard-fs-1" type="state-aware">
    <fileStoreDir>shard-fs-1</fileStoreDir>
  </provider>

  <provider id="shard-fs-2" type="state-aware">
    <fileStoreDir>shard-fs-2</fileStoreDir>
  </provider>
</config>

```

The double-shards template uses a cached provider with two mounts and a redundancy of 1, i.e. only one copy of each artifact is stored.

```

<chain>
  <provider id="cache-fs" type="cache-fs">
    <provider id="sharding" type="sharding">
      <redundancy>1</redundancy>
      <sub-provider id="shard-fs-1" type="state-aware"/>
      <sub-provider id="shard-fs-2" type="state-aware"/>
    </provider>
  </provider>
</chain>

```

To modify the parameters of the template, you can change the values of the elements in the template definition. For example, to increase redundancy of the configuration to 2, you only need to modify the `<redundancy>` tag as shown below.

```

<chain>
  <provider id="cache-fs" type="cache-fs">
    <provider id="sharding" type="sharding">
      <redundancy>2</redundancy>
      <sub-provider id="shard-fs-1" type="state-aware"/>
      <sub-provider id="shard-fs-2" type="state-aware"/>
    </provider>
  </provider>
</chain>

```

Using Balancing to Recover from Mount Failure

In case of a mount failure, the actual redundancy in your system will be reduced accordingly. In the meantime, binaries continue to be written to the remaining active mounts. Once the malfunctioning mount has been restored, the system needs to rebalance the binaries written to the remaining active mounts to fully restore (i.e. balance) the redundancy configured in the system. Depending on how long the failed mount was inactive, this may involve a significant volume of binaries that now need to be written to the restored mount, which may take significant amount of time. Since restoring the full redundancy is a resource intensive operation, the balancing operation is run in a series of distinct sessions until complete. These are automatically invoked after a [Garbage Collection](#) process has been run in the system.

Restoring Balance in Unbalanced Redundant Storage Units

In the case of voluntary actions that cause an imbalance the system redundancy, such as when doing a filestore migration, you may manually invoke rebalancing of redundancy using the [Optimize System Storage](#) REST API endpoint. Applying this endpoint raises a flag for Artifactory to run rebalancing following the next Garbage Collection. Note that, to expedite rebalancing, you can invoke garbage collection manually from the Artifactory UI.

Optimizing System Storage

Artifactory REST API provides an endpoint that allows you to raise a flag to indicate that Artifactory should invoke balancing between redundant storage units of a sharded filestore after the next garbage collection. For details, please refer to [Optimize System Storage](#).

Filtered Resources

Overview

The Filtered Resources Add-on (introduced in Artifactory version 2.3.3) allows treating any textual file as a filtered resource by processing it as a [FreeMarker](#) template.

Each file artifact can be marked as 'filtered' and upon receiving a download request, the content of the artifact is passed through a FreeMarker processor before being returned to the user.

This is an extremely powerful and flexible feature because Artifactory applies some of its own APIs to the filtering context (see below), allowing you to create and provision dynamic content based on information stored in Artifactory.

For example, you can provision different content based on the user's originating IP address or based on changing property values attached to the artifact.

Page Contents

- [Overview](#)
- [Marking an Artifact as a Filtered Resource](#)
- [Filtering Context](#)
- [Provisioning Build Tool Settings](#)
- [Example](#)

Marking an Artifact as a Filtered Resource

Any artifact can be specified as filtered by selecting it in the **Artifact Repository Browser** and setting the **Filtered** checkbox in the **General** tab.

Permissions

You must have **Annotate** permissions on the selected artifact in order to specify it as "Filtered".

The screenshot shows the 'Artifact Repository Browser' interface. On the left, a tree view shows the repository structure, with 'ivy-1.0-local-20120928001044.xml' selected. The main panel displays the details for this artifact under the 'Info' tab. The 'Info' section includes fields for Name, Repository Path, Created, Deployed by, Size, Last Modified, Module ID, Licenses, and Downloaded. A 'Filtered' checkbox is checked. Below the info is the 'Dependency Declaration' section, which shows a code editor with the following XML snippet:

```

1 <dependency org="org.apache.ivy.example" name="list" rev="1.0-local-20120928001044">
2   <artifact name="list" type="ivy" ext="xml"/>
3 </dependency>

```

Filtering Context

Artifactory provides the following environment variables for the FreeMarker template:

- **"properties"** (*org.artifactory.md.Properties*) - Contains the **properties** of the requested artifact and any matrix params included in the request; when a clash of properties with identical keys occurs, the former takes precedence
- **"request"** (*org.artifactory.request.Request*) - The current request that was sent for the artifact
- **"security"** (*org.artifactory.security.Security*) - Artifactory's current security object

Provisioning Build Tool Settings

When logged-in as an admin user, you can provision your user-generated settings for the various build tools (Maven, Gradle and Ivy) using the Filtered Resources features.

To provision user-generated settings:

1. In the **Artifact Repository Browser**, click "Set Me Up" to display the settings generator.
2. Select your build tool, set the appropriate repositories and click "Generate Settings".

The screenshot shows the 'Set Me Up' settings generator interface. It features a form with the following fields:

- Tool:** Maven
- Releases:** libs-release
- Snapshots:** libs-snapshot
- Plugin Releases:** plugins-release
- Plugin Snapshots:** plugins-snapshot
- Mirror Any:**
- remote-repos:** remote-repos

There are buttons for 'Back to Set Me Up', 'Generate Settings', and 'Download snippet'. Below the buttons is a code editor showing the generated XML snippet:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <settings xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.1.0
3   http://maven.apache.org/xsd/settings-1.1.0.xsd" xmlns="http://maven.apache.org/SETTINGS/1.1.0"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
5   <servers>

```

3. Download the generated settings and edit them as required.
4. Back in the **Artifact Repository Browser**, click "Deploy".
5. In the Deploy dialog, set your **Target Repository**, upload your settings file and set your **Target Path**.
6. Click "Deploy" to deploy your settings.

Deploy

Target Repository

ivy-local

Repository-Type: Ivy

Single Multi

settings.xml

Target Path

/settings.xml

Deploy

Example

The following example demonstrates provisioning a different resource based on the current user group and a property on the requested artifact.

In this example, the artifact `'vcsProj.conf.xml'` has a property `'vcs.rootUrl'` which holds the root URL for the version control system. Depending on the user group a different project version control URL is returned.

For the template of `'vcsProj.conf.xml'`:

```
<servers>
<#list properties.get("vcs.rootUrl") as vcsUrl>
  <#list security.getCurrentUserGroupNames() as groupName>
    <vcs>${vcsUrl}/<#if groupName == "dev-product1">product1<#elseif
groupName == "dev-product2">product2<#else>global</#if></vcs>
  </#list>
</#list>
</servers>
```

If, for example, the value of the `'vcs.rootUrl'` property on the `'vcsProj.conf.xml'` artifact is `'http://vcs.company.com'` and the file is downloaded by a developer belonging to the `'dev-product2'` group, then the returned content is:

```
<servers>
  <vcs> http://vcs.company.com/product2 </vcs>
</servers>
```

PGP Signing

Overview

Artifactory lets you manage a pair of GPG signing keys so you can sign packages for authentication in several formats such as Debian, Opkg and YUM. You can manage your GPG signing keys in the **Admin** module under **Security | Signing Keys**.

Generating Keys

The way to generate keys is platform dependent.

The example below shows how to generate the public and private keys on Linux:

Page Contents

- [Overview](#)
 - [Generating Keys](#)
 - [Uploading Keys](#)
 - [Downloading the Public Key](#)

Generating PGP keys

```
# generate the keys
gpg --gen-key

# list all keys in your system and select the pair you want to use in
Artifactory
gpg --list-keys

# resolve the key-id from the lists-keys by selecting the relevant license
pub  2048R/8D463A47 2015-01-19
uid  JonSmith (Jon) <jon.smith@jfrog.com>
key-id =  8D463A47

#export the private key with the specified id to a file
gpg --output {private key file name and path} --armor --export-secret-keys
{key-id}

#export the public key with the specified id to a file
gpg --output {public key file name and path} --armor --export {key-id}
```

You also need to specify a pass phrase that must be used together with the signing keys. The pass phrase can be saved, or passed in with a REST API call.

Uploading Keys

To upload your signing keys, in the **Admin** tab, go to **Security | Signing Keys**.

Signing Keys Management

Manage GPG Signing Keys

Public key : No public key installed

Drop file here or Select file

Private key : No private key installed

Drop file here or Select file

Pass Phrase

Pass-phrase

Once you have specified the key file, select the "Upload" button for the corresponding field.

Artifactory will indicate when keys are installed, and you can click on the **Public key is installed** link to download the public key.

If your signing keys were created with a pass-phrase, enter it in the designated field. You can click "Verify" to make sure the pass-phrase matches the uploaded keys.

Click "Save" to save your changes.

Don't forget to click "Save"
To ensure that your signing keys are properly stored in Artifactory's database, you need to click "Save" even if your signing keys do not have a pass-phrase.

Upload your pass-phrase with REST
If you prefer not to upload your pass phrase using the UI, you can set it using the [REST API](#).

Downloading the Public Key

Once you have uploaded your signing keys, you can download your public key whenever needed using the **Public key is installed** link.

Manage GPG Signing Keys

Public key : **Public key is installed**

Drop file here or Select file

Google Cloud Storage

Overview

From version 4.6, Artifactory fully supports Google Cloud Storage (GCS) so your Artifactory filestore can reside on the cloud. This presents several benefits:

1. **Unlimited scalability**

Since your files are now stored on the cloud, this means that your Artifactory filestore is scalable and effectively unlimited (to the extent offered by your storage provider). You may freely continue to upload files without having to install or maintain any file storage devices. You can even upload files larger than 5 GB using multi-part upload.

2. **Security**

Google's security model offers end-to-end process offering a replicated strategy with all data encrypted both in-flight and at rest.

3. **Disaster recovery**

Since your files are replicated and stored with redundancy, this offers the capability for disaster recovery.

Support for GCS is included with **JFrog Enterprise Edition**.

In order to use GCS with Artifactory, make sure you first [install](#) or [upgrade to](#) Artifactory V4.6 or later.

Backup your system. Your current filestore will be deleted.

Setting up Artifactory to use GCS will delete all files in your current filestore.

If you already have a running installation of Artifactory, then before you setup Artifactory to use GCS and migrate your filestore to the cloud, we strongly recommend that you do a [complete system backup](#).

 Requires an **Enterprise license**

Page Contents

- Overview
- [Setting up Artifactory to Use GCS](#)
 - [Setting Your License](#)
 - [Configuring Artifactory to Use GCS](#)
 - [Interoperable Storage Access Keys](#)
- [Migrating Your Filestore](#)

Setting up Artifactory to Use GCS

First time installation or upgrade

If you are moving your filestore to GCS in the context of upgrading Artifactory, or a first time installation, we recommend that you first do a standard installation of Artifactory using the default settings, or a standard upgrade using your current settings.

In order to move your Artifactory filestore to the cloud, you need to execute the following steps:

- Shut down Artifactory.
- [Set your enterprise license](#)
- [Configure Artifactory to use GCS](#)
- [Migrate your files to the cloud](#)
- Start up Artifactory

Setting Your License

To use Artifactory's support for GCS, you need to have an enterprise license with your Artifactory installation.

To do so, make sure your `$ARTIFACTORY_HOME/etc/artifactory.lic` file contains your enterprise license.

Configuring Artifactory to Use GCS

To configure Artifactory to use a GCS object storage provider, you need to use the Google Cloud Storage binary provider described in [Configuring the Filestore](#) making sure to set the following parameters:

JetS3t Framework

Artifactory uses the JetS3t framework to access GCS. Some of the parameters below are used to set the corresponding value in the framework. For more details, please refer to the [JetS3t Configuration Guide](#).

Parameter	Description
testConnection	Default: true When true, the Artifactory uploads and downloads a file when starting up to verify that the connection to the cloud storage provider is fully functional.
multiPartLimit	Default: 100,000,000 bytes File size threshold over which file uploads are chunked and multi-threaded.
identity	Your cloud storage provider identity.
credential	Your cloud storage provider authentication credential.
bucketName	Your globally unique bucket name on GCS.
path	The relative path to your files within the bucket
proxyIdentity	Corresponding parameters if you are accessing the cloud storage provider through a proxy server.
proxyCredential	
proxyPort	
proxyHost	
port	Default: 80 The port number through which you want to access GCS. You should only use the default value unless you need to contact a different endpoint for testing purposes.
endpoint	Default: <code>commondatastorage.googleapis.com</code> . The GCS hostname. You should only use the default value unless you need to contact a different endpoint for testing purposes.
httpsOnly	Default: false. Set to true if you only want to access GCS through a secure https connection.
httpsPort	Default: 443 The port number through which you want to access GCS securely through https. You should only use the default value unless you need to contact a different endpoint for testing purposes.
bucketExists	Default: false. Only available on google-storage . When true, it indicates to the binary provider that a bucket already exists in Google Cloud Storage and therefore does not need to be created.

The following snippet shows the default chain that uses google-storage as the binary provider:

```
<config version="v1">
  <chain>
    <provider id="cache-fs" type="cache-fs">
      <provider id="eventual" type="eventual">
        <provider id="retry" type="retry">
          <provider id="google-storage" type="google-storage"/>
        </provider>
      </provider>
    </provider>
  </chain>

  <!-- Here is an example configuration part for the google-storage: -->
  <provider id="google-storage" type="google-storage">
    <endpoint>commondatastorage.googleapis.com</endpoint>
    <bucketName><NAME></bucketName>
    <identity>XXXXXXX</identity>
    <credential>XXXXXXXX</credential>
  </provider>
</config>
```

Interoperable Storage Access Keys

The Interoperability API lets you use HMAC authentication and lets GCS interoperate with tools written for other cloud storage systems. To use GCS, you need to turn on this API and use interoperability access details of the current user in GCS. This API is enabled per project member, not per project. Each member can set a default project and maintain their own access keys. For more details, please refer to [Google Cloud Storage Interoperability](#).

You can obtain your access key parameters through your Google GCS account console and set them into the corresponding parameters in Artifactory as follows:

<i>identity</i>	This parameter is provided by GCS as your access key
<i>credential</i>	This parameter is provided by GCS as your access secret

Keep your database settings

Make sure you don't change your database settings in your *db.properties* file.

Migrating Your Filestore

To migrate your filestore, you need to execute the following steps:

- Stop Artifactory
- Copy the `$ARTIFACTORY_HOME/data/filestore` directory to your GCS bucket name and path specified when you configured Artifactory to use GCS.
- Start Artifactory

LDAP Groups

Overview

The LDAP Groups Add-on allows you to synchronize your LDAP groups with Artifactory and leverage your existing organizational structure for managing group-based permissions.

Unlike many LDAP integrations, LDAP groups in Artifactory use super-fast caching, and has support for both Static, Dynamic and Hierarchical mapping strategies. Powerful management is accomplished with multiple switchable LDAP settings and visual feedback about the up-to-date status of groups and users coming from LDAP.

LDAP groups synchronization works by instructing Artifactory about the external groups authenticated users belong to. Once logged-in, you are automatically associated with your LDAP groups and inherit group-based permission managed in Artifactory.

Make sure users log in

Synchronizing LDAP groups does not automatically create users that are members of those groups. Once the LDAP connection is configured, the LDAP users are only created in Artifactory after they log in to Artifactory for the first time. Automatic creation of users can be controlled by the [Auto Create Artifactory Users](#) checkbox in the [LDAP Settings](#) screen.

Page Contents

- [Overview](#)
- [Usage](#)
 - [Group Synchronization Strategies](#)
- [Synchronizing LDAP Groups with Artifactory](#)
 - [Importing Groups Through the UI](#)
 - [Using the REST API](#)
- [Watch the Screencast](#)

Usage

LDAP Groups settings are available in the **Admin** module under **Security | LDAP**.

To use LDAP groups you must first [set up an LDAP server for authentication](#) from the LDAP Settings screen. You must also alert Artifactory about the correct LDAP group settings to use with your existing LDAP schema.

Active Directory Users

For specific help with setting up LDAP groups for an Active Directory installation please see [Managing Security with Active Directory](#).

New LDAP Group Setting

LDAP Group Settings

Settings Name * LDAP Setting

Mapping Strategy **Static** Dynamic Hierarchy

Group Member Attribute * Group Name Attribute *

Description Attribute * Filter *

Search Base Sub-tree Search

Synchronize LDAP Groups

Filter by Username Refresh Import

Group Name	Description	Sync State

Cancel

Group Synchronization Strategies

Artifactory supports three ways of mapping groups to LDAP schemas:

- Static:** Group objects are aware of their members, however, the users are not aware of the groups they belong to. Each group object such as `groupOfNames` or `groupOfUniqueNames` holds its respective member attributes, typically `member` or `uniqueMember`, which is a user DN.
- Dynamic:** User objects are aware of what groups they belong to, but the group objects are not aware of their members. Each user object contains a custom attribute, such as `group`, that holds the group DNs or group names of which the user is a member.
- Hierarchy:** The user's DN is indicative of the groups the user belongs to by using group names as part of user DN hierarchy. Each user DN contains a list of `ou`'s or custom attributes that make up the group association. For example, `uid=user1,ou=developers,ou=uk,dc=jfrog,dc=org` indicates that `user1` belongs to two groups: `uk` and `developers`.

Using OpenLDAP

When using OpenLDAP, you can't apply the **Dynamic** strategy because the `memberOf` attribute is not defined by default (`memberOf` is an overlay), so Artifactory would not be able to fetch it from the LDAP server.

Synchronizing LDAP Groups with Artifactory

Importing Groups Through the UI

Once you have configured how groups should be retrieved from your LDAP server, you can verify your set up by clicking the `Refresh` button on the `Synchronize LDAP Groups` sub-panel. A list of available LDAP groups is displayed according to your settings.

You are now ready to synchronize/import groups into Artifactory. The groups table allows you to select which groups to import and displays the sync-state for each group:

A group can either be completely new or already existing in Artifactory. If a group already exists in Artifactory it can become outdated (for example, if the group DN has changed) - this is indicated in the table so you can select to re-import it.

Once a group is imported (synced) a new external LDAP group is created in Artifactory with the name of the group.

Once you have imported LDAP groups, you can [Manage Permissions](#) on them as with regular Artifactory groups. Users association to these groups is external and controlled strictly by LDAP.

Make sure that LDAP group settings is enabled (in the [LDAP Groups Settings](#) panel) in order for your settings to become effective.

To synchronize a group through the UI, in the **Admin** module, under **Security | LDAP**, select the group you want to synchronize, and search for groups that have been defined under the corresponding group settings. Once groups have been found, select **Import**.

Synchronize LDAP Groups

Search Group by Username (leave blank for *)

2 Records (2 Selected)

Filter by Group Name

Import Page 1 of 1

✓	Group Name	Description	Sync Stat...
✓	testgroup		✓
✓	admins		✓

Import

Once the groups are synchronized, you should see them in your list of groups (**Admin** module under **Security | Groups**) indicated as "External".

Groups Management

+ New

3 Groups

Filter by Group Name

Delete Page 1 of 1

⊙	Group Name	Permissions	External	Auto Join
⊙	admins	-	✓	
⊙	readers	1 Anything		✓
⊙	testgroup	-	✓	

Using the REST API

You may also synchronize LDAP groups by using the [Create Group](#) REST API to create groups with the 'ldap' realm and full DN path to the group object under your LDAP server.

Limitation

Make sure to use lower case only when creating LDAP groups through the REST API. Using upper or mixed case will prevent synchronization of groups.

When using the REST API to synchronize LDAP groups, you need to specify the exact and full Group DN path to the group on your LDAP server. The example below shows the JSON payload you would use to synchronize the "testgroup" group displayed in the below LDAP server:

The screenshot shows the LDAP Browser interface. On the left is a tree view of the directory structure. The right pane shows the details for the group `cn=testgroup,ou=support,ou=UserGroups,dc=openstack,dc=org`. The bottom pane shows a log entry for a successful modification.

Attribute Description	Value
<i>objectClass</i>	<i>top (abstract)</i>
<i>objectClass</i>	<i>groupOfNames (structural)</i>
<i>cn</i>	testarou
<i>member</i>	cn=Larry Cai,ou=Users,dc=openstack,dc=org

```

#!RESULT OK
#!CONNECTION ldap://192.168.59.128:389
#!DATE 2016-08-23T21:06:45.783
dn: cn=testGroup,ou=il,ou=support,ou=UserGroups,dc=openstack,dc=org
changetype: modify
replace: member
member: cn=Larry Cai,ou=Users,dc=openstack,dc=org
-

```

Sample JSON:

```

{
  "name": "testgroup",
  "description" : "This groups already exists in ldap",
  "autoJoin" : false,
  "realm": "ldap",
  "realmAttributes":
  "ldapGroupName=testgroup;groupsStrategy=STATIC;groupDn=cn=testgroup,ou=support,ou=UserGroups,dc=openstack,dc=org"
}

```

Watch the Screencast

License Control

Controlling Third Party Licenses

The License Control Add-on completes the [Artifactory Build Integration Add-on](#) allowing you full control over the licenses of the dependencies used by your builds (and eventually in your software).

This Add-on is part of the Artifactory Pro Power Pack.

As part of the Build Server deployment to Artifactory, it analyzes the used dependencies and tries to match them against a set of license management rules.

Notifications can be sent to a selected list of recipients about dependencies with unknown or unapproved license information.

To support this feature Artifactory includes a new license management facility where rules about license

matching and approval status are defined. These rules are consulted as part of the license analysis.

How does license analysis work?

Automatic analysis is performed upon deployment by examining information found in artifact module files. Currently **Maven POM, Ivy Descriptor, NuGet, and RPM** files are supported.

You can always override the automatic results and assign license information manually to dependencies. You can also compare the current license status to the auto calculated one and decide what results of the automatic analysis to accept.

License information is stored with the artifact and reused by the automatic license analysis on subsequent builds.

Page Contents

- Controlling Third Party Licenses
- Central License Management
 - Editing License Information
- Using Build Licenses
 - Build Server Configuration
 - Examining Build Licenses
 - Running Manual License Discovery
- Setting License Information Manually
- Licenses REST API

Central License Management

Licenses are managed in the **Admin** module under **Configuration | Licenses**.

License Key	Name	URL	Status
AFL-3.0	The Academic Free License 3.0	http://www.opensource.org/licenses/afl-3.0.php	Approved
AGPL-V3	GNU AFFERO GENERAL PUBLIC LICENSE v3	http://www.opensource.org/licenses/agpl-v3.html	Unapproved
Apache-1.0	The Apache Software License, Version 1.0	http://apache.org/licenses/LICENSE-1.0	Unapproved
Apache-1.1	The Apache Software License, Version 1.1	http://apache.org/licenses/LICENSE-1.1	Approved
Apache-2.0	The Apache Software License, Version 2.0	http://www.opensource.org/licenses/apache2.0.php	Unapproved
APL-1.0	Adaptive Public License 1.0	http://www.opensource.org/licenses/apl1.0.php	Unapproved
APSL-2.0	The Apple Public Source License 2.0	http://www.opensource.org/licenses/apsl-2.0.php	Approved
Artistic-License-2.0	Artistic License 2.0	http://www.opensource.org/licenses/artistic-license-2...	Unapproved
Attribution	The Attribution Assurance License	http://www.opensource.org/licenses/attribution.php	Unapproved
Bouncy-Castle	Bouncy Castle License	http://www.bouncycastle.org/licence.html	Unapproved
BSD	Berkeley Software Distribution (BSD)	http://www.opensource.org/licenses/bsd-license.php	Approved
BSL-1.0	Boost Software License 1.0 (BSL1.0)	http://www.opensource.org/licenses/bsl1.0.html	Approved
CA-TOSL-1.1	Computer Associates Trusted Open Source License 1.1	http://www.opensource.org/licenses/ca-tosl-1.1.php	Unapproved

You can add a new license by clicking **New** or edit the information for a license by selecting its **License Key** in the list.

Artifactory comes preconfigured with all the common **OSI** licenses and JFrog has already tuned these licenses against common project builds.

By selecting **Export**, you can also export the license list and import it later on to new Artifactory instances.

Editing License Information

License Key	A unique identifier for this license in Artifactory
Long Name	A description of the license
URLs	The URL that describes the terms of the license
Notes	Additional notes
RegExp	The regular expression by which to match the license (by comparing it to license information in module files). <div style="border: 1px solid green; padding: 5px; margin: 5px 0;"> <p>If you leave the regexp field blank, Artifactory attempts an exact match against the license key.</p> </div>
Approved	When set, this license is approved which means you allow the use of components that come with this license.

Using Build Licenses

Build Server Configuration

When you run a build from your CI server (Hudson, TeamCity or Bamboo), configure the Artifactory Plugin to run license checks as part of the build.

Below is a sample section from the Hudson configuration of the Artifactory Plugin:

Run License Checks (requires Pro) ?

Send license violation notifications to: ?

Comma-separated list of recipient addresses.

You can configure whether or not you wish license checks to take place as part of deploying Build Info to Artifactory (the Build Info Bill of Materials must be deployed to Artifactory for license checks to run).

You can also set a list of recipients to be notified about license violations as soon as they occur. This way whenever a dependency with an unknown or unapproved license is added to the build recipients receive an immediate email notification and can tend to any potential license violation.

Sending license violation notifications is performed through Artifactory and requires a [valid mail server](#) to be configured.

Not failing the build
Currently, Artifactory does not fail the build as a result of license violations.

This is an informed decision in the spirit of allowing technical development to continue, while alerting others about the advent of unauthorized dependencies in near or real-time, so they can be addressed early on by the appropriate parties.

Examining Build Licenses

Once the build has finished on the build server and Build Info has deployed to Artifactory, license checks are run.

You can view detailed license information in the **Licenses** tab of the **Build Browser**. This tab displays information about all the dependencies used in the build and the license they are associated with. To group the information by **Scope** or **License** click the corresponding column header.

Build Browser

Build #60

General Build Info | Published Modules | Environment | Issues | **Licenses** | Governance | Diff | >>

Summary: Unapproved: 3 | Not Found: 13 | Unknown: 0 | Neutral: 0 | Approved: 2

Includes

- Include Published Artifacts
- Include dependencies of the following scopes:
 - compile
 - test
 - provided
 - runtime

Export to CSV | Auto-find Licenses

Filter by Artifact ID

Artifact ID	Scopes	Repo Path	License
aopalliance:aopalliance:1.0	compile	gradle-libs-cache:aopalliance/aopalliance/1.0...	Public Domain
org.springframework:spring-beans:2.5.6	compile	Not in repository (externally resolved or dele...	Not Found
org.springframework:spring-aop:2.5.6	compile	Not in repository (externally resolved or dele...	Not Found
org.springframework:spring-core:2.5.6	compile	Not in repository (externally resolved or dele...	Not Found
org.testng:testng:5.9	test	gradle-libs-cache:org/testng/testng/5.9/testn...	Not Found
javax.servlet:servlet-api:2.5	provided	java.net.m1-cache:javax/servlet/servlet-api/2...	Not Found
javax.mail:mail:1.4	compile	gradle-libs-cache:javax/mail/mail/1.4/mail-1...	Not Found

The summary panel displays the overall count of licenses by status and inside the table itself, licenses are displayed in different colors according to their status:

License Status	Description
Unapproved	The license found is not an approved license

Unknown	License information was found but cannot be related to any license managed in Artifactory
Not Found	No license information could be found for the artifact.
Neutral	The license found is unapproved, however another approved license was found for the artifact (Only applicable for artifacts that are associated with multiple licenses)
Approved	The license found is an approved license

Inline License Editing

From the Build Browser, an Artifactory administrator can manually change the license information for any artifact displayed. Clicking the entry under the **License** column for any artifact will display the **Edit 'artifactory.licenses' Property** dialog where the administrator can specify the licenses for that artifact. For example, clicking the *Public Domain* license entry from the screenshot above will display the following dialog:

Edit 'artifactory.licenses' Property

The dialog shows a search filter at the top left. Below it is a list of 'Available Predefined Values' including AFL-3.0, AGPL-V3, APL-1.0, Apache-2.0, Apache-1.0, Apache-1.1, APSL-2.0, and Artistic-License-2.0. In the center, there are four navigation buttons: a double left arrow, a single left arrow, a single right arrow, and a double right arrow. To the right, under 'Selected Predefined Values', the 'Public Domain' license is selected. At the bottom right, there are three buttons: 'Clear All', 'save', and 'Cancel'.

Running Manual License Discovery

You can manually run the license discovery rules after a build has already run. There are several reasons why you may want to do this:

1. License rules (configured licenses and regular expressions) have changed and you want to compare the existing build licenses with the results of the new rules, or use them to complete missing license information.
2. To test the current rules against the dependencies and tweak the rules, if necessary.
3. To check which license information can come from rules and which license information must be set manually.

To trigger license discovery select the "Auto-find Licenses" button.

Any license conflicts are displayed to the right of the table. You can override the existing license information with the discovered license by checking the corresponding checkbox (you must have annotated permissions for the artifacts for which you want to override licenses).

All Builds > maven-example > 60

Build Browser

Build #60

General Build Info | Published Modules | Environment | Issues | **Licenses** | Governance | Diff | Release History | Build Info JSON

Includes

- Include Published Artifacts
- Include dependencies of the following scopes:
 - compile
 - test
 - provided
 - runtime

Export to CSV | Auto-find Licenses | **Override Selected Licenses** | Cancel

Filter by Artifact ID < page 1 of 1 >

Artifact ID	Scopes	Repo Path	License	Found Licenses	Override
org.springframework:spring-beans:2.5.6	compile	Not in repository (externally resolved or de...	Not Found	Not Found	<input type="checkbox"/>
org.springframework:spring-aop:2.5.6	compile	Not in repository (externally resolved or de...	Not Found	Not Found	<input type="checkbox"/>
org.springframework:spring-core:2.5.6	compile	Not in repository (externally resolved or de...	Not Found	Not Found	<input type="checkbox"/>
org.testing:testing:5.9	test	gradle-libs-cache/testing/testing/5.9/tes...	Not Found	Apache-2.0	<input type="checkbox"/>
javax.servlet:servlet-api:2.5	provided	java.net.m1-cache/javax/servlet/servlet-api...	Not Found	Not Found	<input type="checkbox"/>
javax.mail:mail:1.4	compile	gradle-libs-cache/javax/mail/mail/1.4/mail...	Not Found	CDL-1.0	<input type="checkbox"/>
javax.servlet.jsp:jsp-api:2.1	compile	gradle-libs-cache/javax/servlet/jsp/jsp-api/...	Not Found	Not Found	<input type="checkbox"/>
aopalliance:aopalliance:1.0	compile	gradle-libs-cache/aopalliance/aopalliance/1...	Public Domain	Public Domain	<input type="checkbox"/>
org.codehaus.plexus:plexus-utils:1.5.1	compile	gradle-libs-cache/org/codehaus/plexus/ple...	Day	Apache-2.0	<input type="checkbox"/>
org.codehaus.plexus:plexus-utils:1.5.1	compile	gradle-libs-cache/org/codehaus/plexus/ple...	Day-Addendum	Apache-2.0	<input type="checkbox"/>
org.codehaus.plexus:plexus-utils:1.5.1	compile	gradle-libs-cache/org/codehaus/plexus/ple...	Apache-2.0	Apache-2.0	<input type="checkbox"/>
org.codehaus.plexus:plexus-utils:1.5.1	compile	gradle-libs-cache/org/codehaus/plexus/ple...	Apache-1.1	Apache-2.0	<input type="checkbox"/>
commons-logging:commons-logging:1.1.1	compile	gradle-libs-cache/commons-logging/comm...	Not Found	Apache-2.0	<input type="checkbox"/>
commons-io:commons-io:1.4	compile	gradle-libs-cache/commons-io/commons-i...	Not Found	Apache-2.0	<input type="checkbox"/>
hsqldb:hsqldb:1.8.0.10	runtime	repo1-cache/hsqldb/hsqldb/1.8.0.10/hsqid...	Not Found	HSQldb	<input type="checkbox"/>
javax.activation:activation:1.1	compile	gradle-libs-cache/javax/activation/activati...	Not Found	CDL-1.0	<input type="checkbox"/>

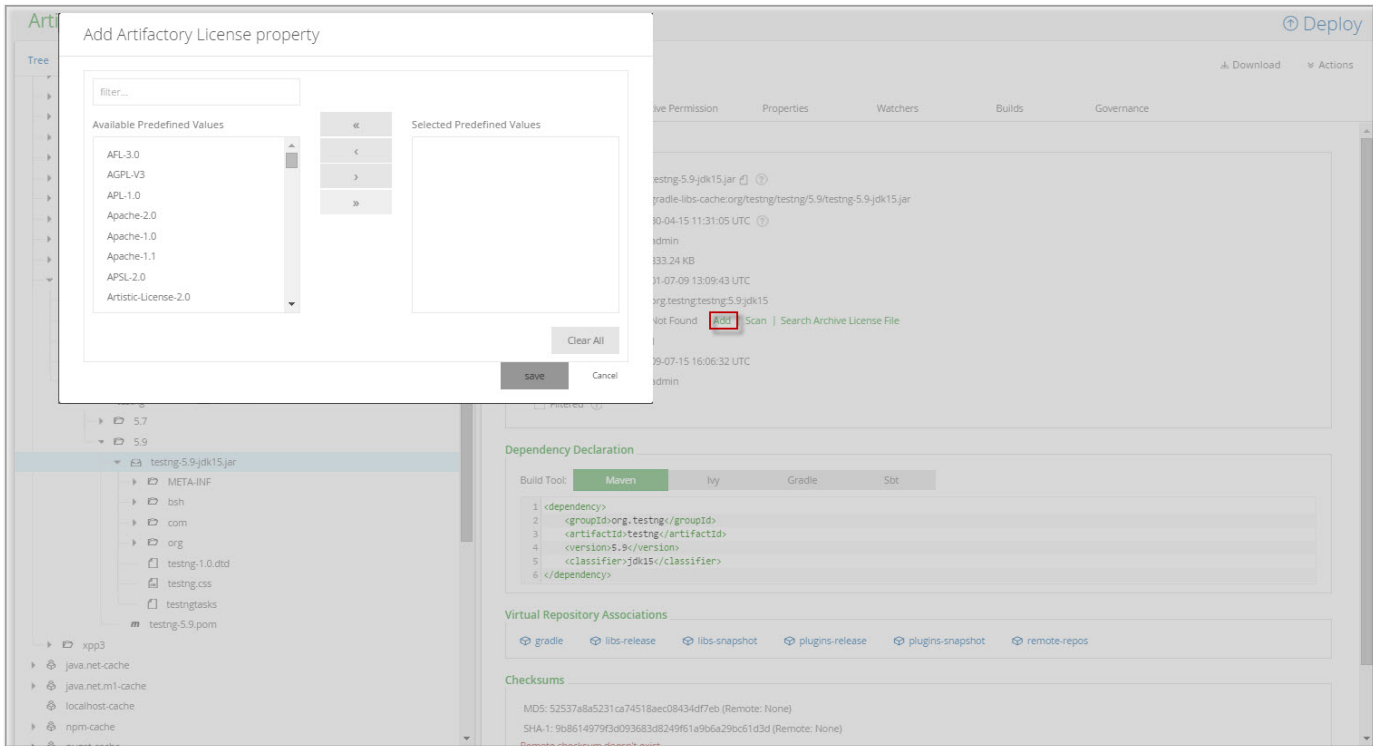
Setting License Information Manually

To set license information for artifacts manually, when viewing an artifact's details in the **Artifact Repository Browser**, in the **General** tab **Licenses** entry, click **Add**.

This will display the **Add Artifactory Licenses Property** dialog where you can specify the licenses for the selected artifact.

Multiple licenses

Note that an artifact may be associated with multiple licenses



Scanning artifact Maven/Ivy model for license

Another option for editing the license information is by scanning the Maven/Ivy model for licenses, that is, looking for an existing pom matching the artifact.

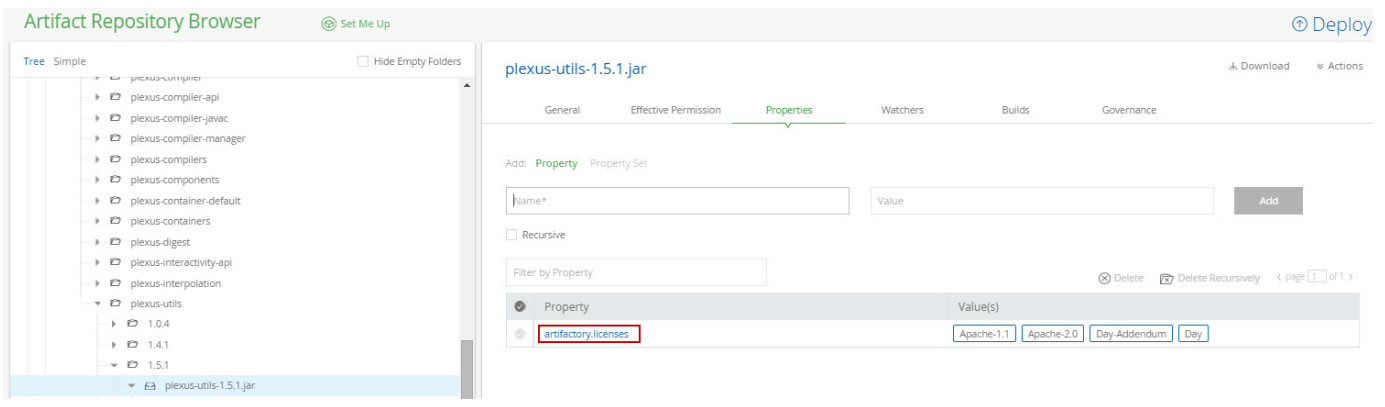
Once you have the artifact selected in the tree browser go to the **General** tab and under the **License** label choose **Scan** and confirm licenses found in the scan results, if any.

Yet another option would be to use the 'Search For Archive License File' link, which will scan the artifact archive for a 'License' or 'License.txt' entry and ask for confirmation, if found.

License Information as Properties

Internally, license information is stored as regular **properties**, using the built-in `artifactory.licenses` property name.

Therefore, all operations with properties are available to license information (searches, recursive assignment, property-based deployment and resolution etc.)



Licenses REST API

License-oriented searches and management operations are available through the REST API.

Refer to the [REST API Documentation](#) for usage information.

OAuth Integration

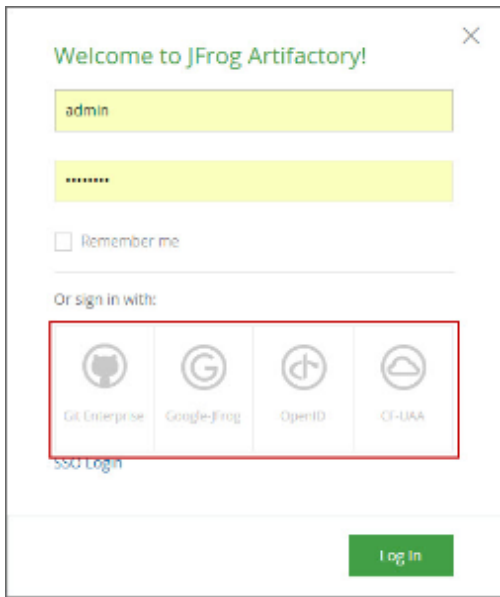
Overview

From version 4.2, Artifactory is integrated with OAuth allowing you to delegate authentication requests to external providers and let users login to Artifactory using their accounts with those providers.

Currently, the provider types supported are **Google**, **OpenID Connect**, **GitHub Enterprise**, and **Cloud Foundry UAA**. You may define as many providers of each type as you need.

Usage

When OAuth is enabled in Artifactory, users may choose to sign in through any of the supported OAuth providers. To log in through a provider, simply click on the provider's button in the login screen.



You will be redirected to the login screen of the corresponding provider.

If you are already logged in to any of that provider's applications you will not need to log in again, but you may have to authorize Artifactory to access your account information, depending on the provider type.

Page Contents

- [Overview](#)
- [Usage](#)
- [Configuring OAuth](#)
 - [Adding a New Provider](#)
 - [Using Query Params](#)
- [Binding Existing User Accounts](#)
- [Creating OAuth Provider Accounts](#)
 - [GitHub OAuth Setup](#)
 - [Google OAuth Provider Setup](#)
 - [Cloud Foundry UAA Setup](#)
- [Using Secure OAuth](#)
- [Using API Key with OAuth Users](#)
- [Using OAuth on High Availability Setup](#)

Configuring OAuth

To access OAuth integration settings, in the **Admin** module, select **Security | OAuth SSO**.

OAuth SSO Configuration

General OAuth Settings

- Enable OAuth
- Auto Create Artifactory Users
- Allow Users Access To Profile Page

Default GitHub Provider

Git Enterprise

Reset

Save

Enable OAuth	If checked, authentication with an OAuth provider is enabled and Artifactory will display all OAuth providers configured. If not checked, authentication is by Artifactory user/password.
Auto Create Artifactory Users	If checked, Artifactory will create an Artifactory user account for any new user logging in to Artifactory for the first time.
Default Provider	Specifies the provider through which different clients (such as NPM, for example) should authenticate their login to gain access to Artifactory. <div style="border: 1px solid orange; padding: 5px;">Default provider Currently, only a GitHub Enterprise OAuth provider may be defined as the Default Provider.</div>
Allow Created Users Access To Profile Page	When checked, users created after authenticating using OAuth, will be able to access their profile . This means they are able to generate their API Key and set their password for future use.

Custom URL base

For your OAuth settings to work, make sure you have your [Custom URL Base](#) configured.

Adding a New Provider

The list of providers defined in Artifactory is displayed in the **Providers** section.

Providers

4 Providers ⊕ New

Filter by Name

Page 1 of 1

Name	Type	ID	Auth Url	Enabled
CF-UAA	Cloud Foundry	login	https://jfrog.com/artifactory/uaa	<input checked="" type="checkbox"/>
Git Enterprise	GitHub	https://github.com	https://github.com/login/oauth/authorize	<input checked="" type="checkbox"/>
Google-JFrog	Google	JFrog	https://google.com	<input checked="" type="checkbox"/>
OpenID	OpenID	admin	https://google.com	<input checked="" type="checkbox"/>

To add a new provider, click "New". Artifactory displays a dialog letting you enter the provider details. These may vary slightly depending on the provider you are configuring.

Create New Provider

Provider Settings

Enabled

Provider Name *

Provider Type *

Provider ID *

Secret *

Basic URL *








Auth URL *




API URL *

Token URL *

The following table describes the settings required by each supported provider, and the corresponding values you should use (where available):

	<i>GitHub.com</i>	<i>GitHub Enterprise</i>	<i>Google</i>

Enabled			
Provider Name			
Provider Type			
Provider ID			
Secret			
Domain			
Docker Login			

Npm Login			
Basic URL	<code>https://github.com/</code>	<code><Server Base URL></code>	
Auth URL	<code>https://github.com/login/oauth/authorize</code> <div style="border: 1px solid orange; padding: 5px; margin-top: 10px;"> <p>GitHub.com Accounts Any GitHub.com account that has access to the Artifactory URL will be allowed to login, including accounts that are outside your GitHub.com organization scope.</p> </div>	<code><Server Base URL>/login/oauth/authorize</code>	<code>https://accounts.google.com/o/oauth2/authorize</code>
API URL	<code>https://api.github.com/user</code>	<code><Server Base URL>/api/v3/user</code>	<code>https://www.googleapis.com/oauth2/</code>
Token URL	<code>https://github.com/login/oauth/access_token</code>	<code><Server Base URL>/login/oauth/access_token</code>	<code>https://www.googleapis.com/oauth2/</code>

Using Query Params

You may pass query params along with the [Authorization URL](#). For example,

```
https://github.com/login/oauth/authorize?realm=Employees
```

Multiple query params should be separated with an ampersand. For example,

```
https://github.com/login/oauth/authorize?realm=Employees?client_id=XXXXXXXXXXXX&scope=openid%20profile%20email
```





Binding Existing User Accounts

If you already have an **internal (not external realms such as LDAP, SAML...)** account in Artifactory, in order to be able to login using any of your OAuth provider accounts, you need to bind your Artifactory account to the corresponding account.

To bind your account, go to your [Profile](#) page and enter your Artifactory password to unlock it.

Under **OAuth User Binding**, select **Click to bind** next to the OAuth provider you wish to bind to.

OAuth User Binding

 Git Enterprise	Click to bind
 Google-JFrog	Click to bind
 OpenID	Click to bind
 CF-UAA	Click to bind

Creating OAuth Provider Accounts

In order to use OAuth authentication, you need to set up an account with each OAuth provider you wish to use in order to get the various parameters (such as Provider ID and Secret) you will need to set up OAuth integration in Artifactory.

GitHub OAuth Setup

Caution: Access to GitHub.com Accounts

Any [GitHub.com](#) account that has access to the Artifactory URL will be allowed to login, including accounts that are outside your [GitHub.com](#) organization scope. This does not apply to GitHub Enterprise.

To set up your OAuth account on GitHub, execute the following steps:

1. Login to your GitHub account. Under your personal profile settings, select **Applications** and click the [Developer Applications](#) tab.
2. Click **Register new application**.
3. Set the **Application name**. For example, **Artifactory SaaS OAuth**.
4. Set the **Homepage Url**. This is your Artifactory server host URL (`https://<artifactory-server>/`).
For example, `https://mycompany.jfrog.io/mycompany/`
5. Set the **Authorization Callback Url** as follows:
 - a. For Artifactory on-prem installation: `http://<server_host>/artifactory/api/oauth2/loginResponse`
For example, `http://mycompany.artifactory.com/artifactory/api/oauth2/loginResponse`
 - b. For Artifactory SaaS: `https://<server_name>.jfrog.io/<server_name>/api/oauth2/loginResponse`
For example, `https://mycompany.jfrog.io/mycompany/api/oauth2/loginResponse`
6. Click **Register application** to generate your **Client ID** and **Client Secret**.
Make a note of these; you will need them to configure OAuth authentication through GitHub on Artifactory.

Authorized applications **Developer applications**

Register a new OAuth application

Application name

 Something users will recognize and trust

Homepage URL


 The full URL to your application homepage

Application description

 This is displayed to all potential users of your application

Authorization callback URL


 Your application's callback URL. Read our [OAuth documentation](#) for more information


 Drag & drop
 or [choose an image](#)

Google OAuth Provider Setup

To set up your OAuth account on Google, execute the following steps:

1. Login to [Google Developer Console](#).
2. Create a new project. For example, "Artifactory OAuth".
3. Once the project is created, in the left navigation bar, select **APIs & auth | Credentials**.
4. Select the **OAuth consent screen** tab and configure the consent screen end users will see when logging in with the Google credentials.

Google Developers Console Artifactory OAuth [Sign up for a free trial.](#) [Try the beta console](#) 

Home

Permissions

APIs & auth

APIs

Credentials

Monitoring

Source Code

Cloud Launcher

Deployments

Compute

Networking


Storage

Big Data

Credentials **OAuth consent screen** Domain verification


The consent screen will be shown to users whenever you request access to their private data using your client ID


Note: This screen will be shown for all applications using this project's OAuth 2.0 client IDs

Email address 

Product name

Homepage URL (Optional)



Product logo URL (Optional) 







This is how your logo will look to end users
Max size: 120x120 px

Privacy policy URL (Optional)

Terms of service URL (Optional)

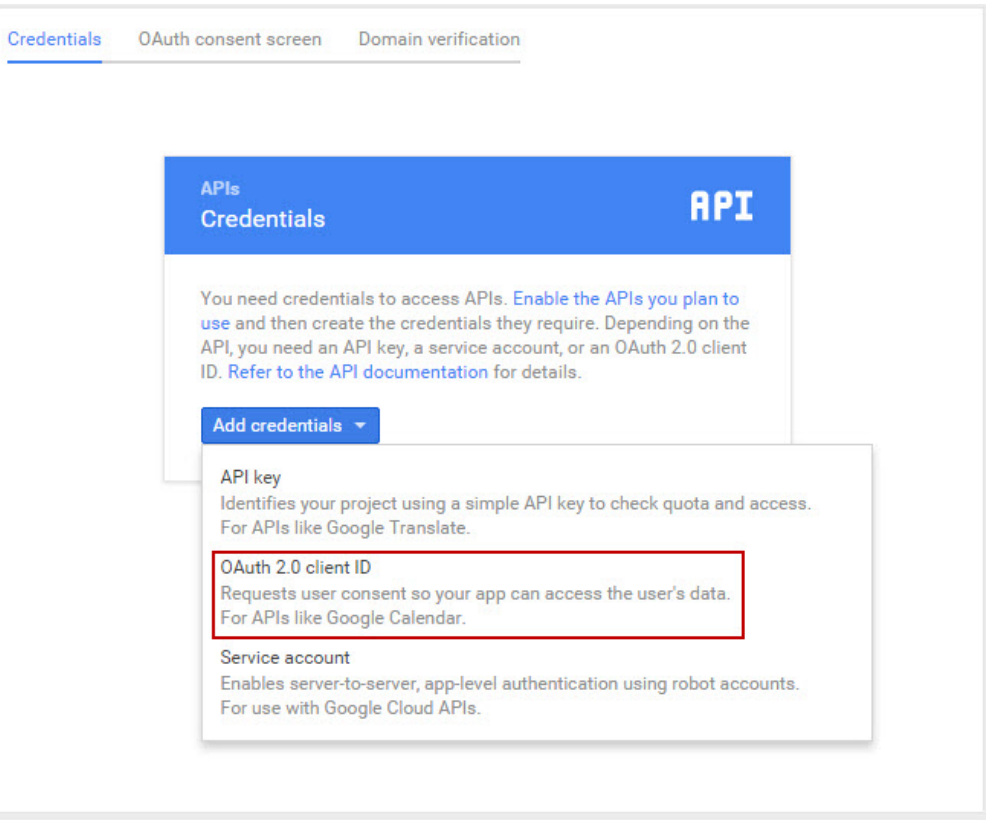
 
 Logo

- Project Name would like to:

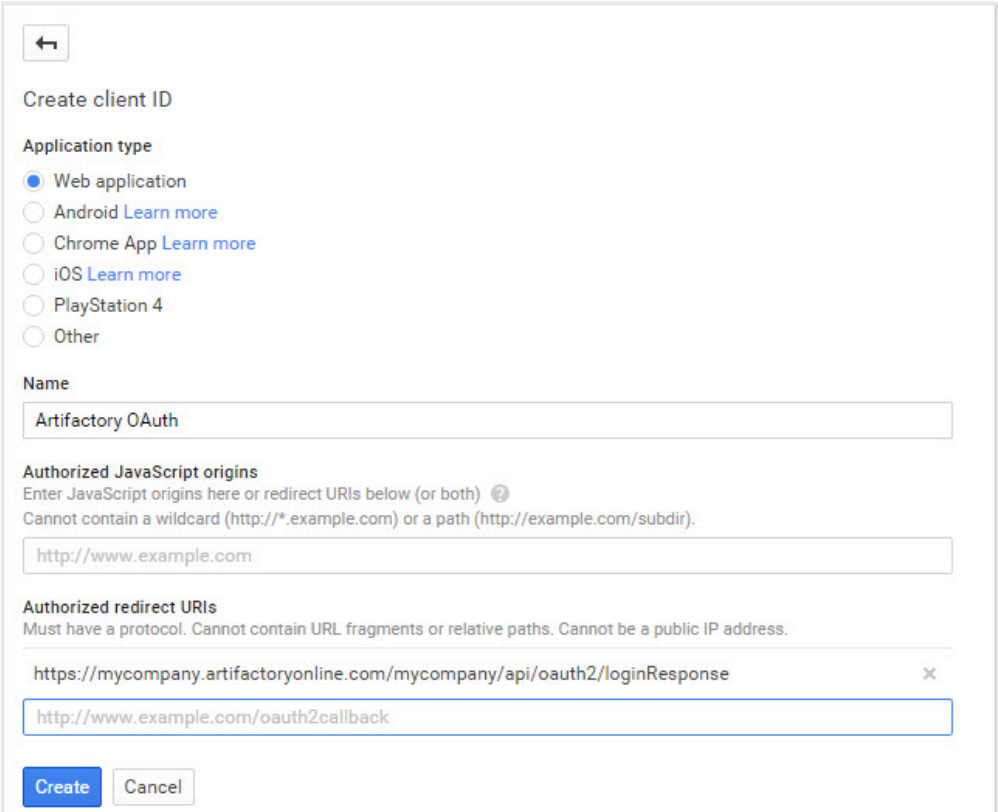
-  Know your basic profile info and list of people in your circles. 
-  Make your listen, app and comment activity available via Google, visible to: [Your circles](#) 

By clicking **Accept**, you allow this app and Google to use your information in accordance with their respective terms of service and privacy policies. You can change this and other [Account Permissions](#) at any time.

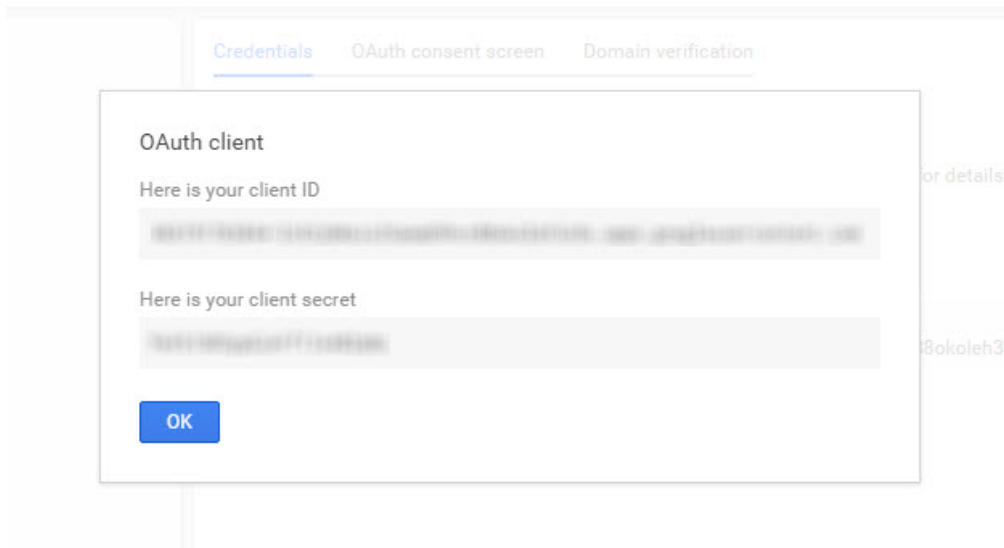
5. Back in the **Credentials** tab, Click **Add Credential** and select **OAuth 2.0 client ID**



- 6. Under **Create client ID**, select **Web application**.
- 7. Enter a **Name** and set the **Authorized redirect URIs**
For Artifactory on-prem: `https://<server_host>/artifactory/api/oauth2/loginResponse`
For Artifactory SaaS: `https://<server_name>.jfrog.io/<server_name>/api/oauth2/loginResponse`



- 8. Click **Create** to generate your **Client ID** and **Client Secret**.



Make a note of these; you will need them to configure OAuth authentication through Google on Artifactory.

Cloud Foundry UAA Setup

OAuth authentication with Cloud Foundry UAA is supported from Artifactory version 4.2.1.

To setup your OAuth authentication with Cloud Foundry UAA, fill in the fields as needed.

UAA	Vm Credentials	
	Admin Credentials	
	Push Console Credentials	
	Run Smoke Tests Credentials	
	System Services Credentials	
	System Verification Credentials	
	System Passwords Client Credentials	
	Login Client Credentials	

Using Secure OAuth

To use secure OAuth with a valid certificate from a CA trusted by Java, all you need to do is use a secure OAuth URL in your settings.

If you want to use OAuth with a non-trusted (self-signed) certificate, please follow the steps described in [Using a Self-Signed Certificate](#).

Using API Key with OAuth Users

While OAuth provides access to Artifactory UI, it is also possible for OAuth users to generate an [API key](#) that can be used instead of a password for basic authentication or in a dedicated [REST API header](#), this is very useful when working with different clients, e.g. docker, npm, maven, etc. or using Artifactory REST API.

In order to allow OAuth users access to an API key you will need to make sure that the "**Auto Create Artifactory Users**" and "**Allow Created Users Access To Profile Page**" check boxes are checked. This means that OAuth users are also saved in Artifactory database and can access their [profile page](#) in order to generate, retrieve and revoke their API key.

Using OAuth on High Availability Setup

The OAuth protocol requires the client to give permission to a specific application. Artifactory will redirect the user to the configured application URL and one permission is granted user will be navigated back.

The limitation on this process when working in High Availability setup is that the user must return to the same node, otherwise the authentication process will fail, in order to achieve this a sticky session configuration should include the `/artifactory/api/oauth2/`.

The example below shows NGINX configuration.

```


NGINX Reverse Proxy Configuration



```
location ~ (/artifactory/webapp/|/artifactory/ui/|/artifactory/api/oauth2/)
{
 proxy_http_version 1.1;
 proxy_pass http://<UPSTREAM_NAME>;
 proxy_intercept_errors on;
 proxy_pass_header Server;
 proxy_connect_timeout 75s;
 proxy_send_timeout 2400s;
 proxy_read_timeout 2400s;
 proxy_set_header Host $host;
 proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
 proxy_set_header X-Forwarded-Proto $http_x_forwarded_proto;
 proxy_set_header X-Real-IP $remote_addr;
 proxy_set_header X-Artifactory-Override-Base-Url
 $http_x_forwarded_proto://$host/artifactory;
}
```


```

Properties

Overview

Artifactory allows you to place properties on both artifacts and folders. Setting (and deleting) properties is supported by local repositories or local-cache repositories. While you cannot set or delete properties on virtual repositories, you can retrieve them.

You can assign properties from the **UI**, via **REST** (see below), or **on deployment**, using [Matrix Parameters](#). Properties can also be used to [Control Artifacts Resolution](#).

Properties are **searchable** and can be combined with [Smart Searches](#) to search for items based on their properties and then manipulate all the items in the search result in one go.

Property Sets

You can define the collections of properties called 'Property Sets' in the user interface. In each property-set you can define properties and for each property specify whether the property is open, single-value or multi-value.

This impacts the user interface you see when setting a property value and when searching for property values. Using searchable properties in artifact management is a very powerful feature.

Page Contents

- [Overview](#)
 - [Property Sets](#)
- [Attaching Properties via the UI](#)
- [Attaching and Reading Properties via REST API](#)

Read More

- [Using Properties in Deployment and Resolution](#)

Properties are for Guiding, not for Restricting

When you define a property-set with 'strongly-typed' property values, those values are used to provide an intuitive, guiding UI for tagging and locating items.

The actual value does not force a strong relationship to the original property-set's predefined values. This is by design, to not slow-down common repository operations and for keeping artifacts management simple by allowing properties to change and evolve freely over time, without worrying about breaking older property rules.

Properties are therefore a helpful and non-restrictive feature.

Attaching Properties via the UI

When selecting any item in the tree browser, you can view its **Properties** tab to view or edit the properties attached to the item.

The screenshot shows the 'antlr-2.7.7.jar' item selected in a tree browser. The 'Properties' tab is active, displaying a form to add new properties and a table of existing ones. The 'Add' form includes a 'Name *' field, a 'Value' field, and an 'Add' button. There is also a 'Recursive' checkbox. The table below shows three existing properties: 'build.name' with value 'genericDeploy', 'build.timestamp' with value '1435677419561', and 'build.number' with value '30'. The table has a 'Filter by Property' input, a 'Delete' button, and a page indicator 'Page 1 of 1'.

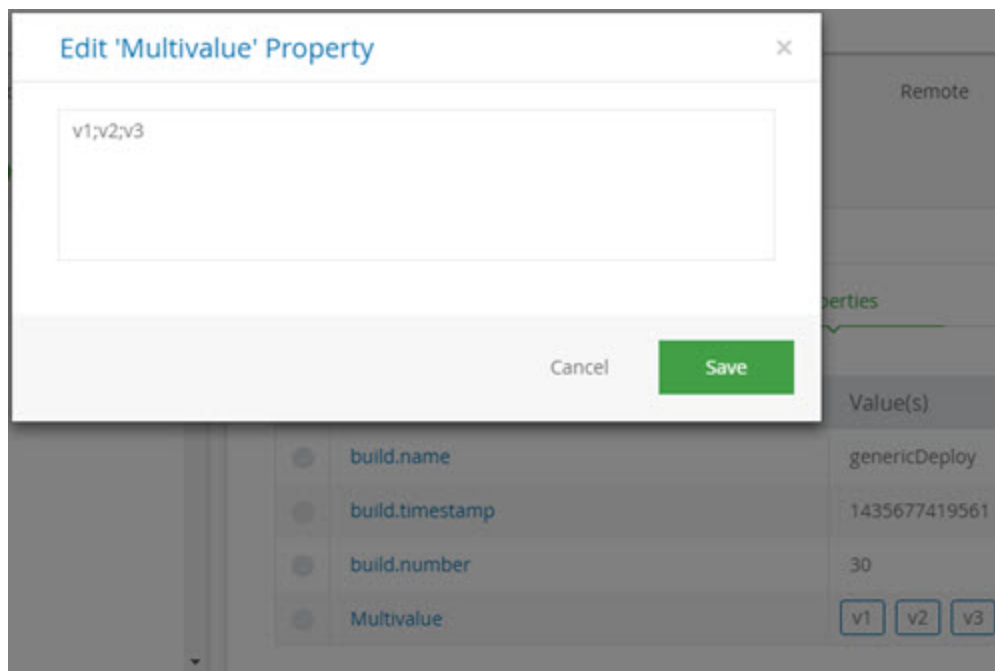
Property	Value(s)
build.name	genericDeploy
build.timestamp	1435677419561
build.number	30

To add a property, simply enter its name and value and click "Add".

To add multi-value properties, enter the values separated with a semi-colon (;). For example:

This screenshot shows the 'Properties' tab with the 'Add' form. The 'Name *' field contains 'Multivalued' and the 'Value' field contains 'v1;v2;v3'. The 'Add' button is highlighted in green.

You can edit the value of any property by clicking on it



Attaching and Reading Properties via REST API

Properties are a special form of metadata and are stored on items just like any metadata - in XML form.

In fact, you can view properties not only from the *Artifacts:Properties* tab, but also from the *Artifacts:Metadata* tab, in which you can examine properties as they are stored in XML form. The properties XML is using the `properties` root tag and has a very simple format.

You can set, retrieve and remove properties from repository items via REST API, as you would do with any other XML-based metadata.

Using Properties in Deployment and Resolution

Introducing Matrix Parameters

Matrix parameters key-value pairs parameters separated by a semicolon (;) that you can place anywhere on a URI.

This is a standard method for specifying parameters in HTTP (in addition to querying parameters and path parameters).

For example:

```
http://repo.jfrog.org/artifactory/libs-releases-local/org/libs-releases-local/org/jfrog/build-info-api/1.3.1/build-info-api-1.3.1.jar;status=DEV;rating=5
```

Artifactory makes use of matrix parameters for:

1. Adding properties to artifacts as part of deployment
2. Controlling artifact resolution using matrix parameters

Page Contents

- [Introducing Matrix Parameters](#)
- [Dynamically Adding Properties to Artifacts on Deployment](#)
- [Controlling Artifact Resolution with Matrix Parameters Queries](#)
 - [Non-mandatory Properties](#)
 - [Mandatory Properties](#)
- [Multi-valued Properties Support](#)

Dynamically Adding Properties to Artifacts on Deployment

You can add key-value matrix parameters to deploy (PUT) requests and those are automatically transformed to properties on the deployed

artifact.

Since matrix parameters can be added on any part of the URL, not just at the end, you can add them to the target deployment base URL. At the time of deployment, the artifact path is added after the matrix parameters and the final deployed artifact will be assigned the defined properties.

You can even use dynamic properties, depending on our deployment framework.

When using Maven, for instance, you can add two parameters to the deployment URL: `buildNumber` and `revision`, which Maven replaces at deployment time with dynamic values from the project properties (e.g. by using the Maven build-number plugin).

So, if you define the distribution URL as:

```
http://myserver:8081/artifactory/qa-releases;buildNumber=${buildNumber};revision=${revision}
```

And deploy to the `qa-releases` repository a jar with the following path:

```
/org/jfrog/build-info-api/1.3.1/build-info-api-1.3.1.jar
```

Upon deployment the URL is transformed to:

```
http://myserver:8081/artifactory/qa-releases;buildNumber=249;revision=1052/org/jfrog/build-info-api/1.3.1/build-info-api-1.3.1.jar
```

And the deployed `build-info-api-1.3.1.jar` has two new properties:

```
buildNumber=249  
revision=1052
```

Permissions to attach properties

You must have the 'Annotate' permission in order to add properties to deployed artifacts.

Controlling Artifact Resolution with Matrix Parameters Queries

Matrix parameters can also be used in artifact resolution, to control how artifacts are found and served.

There is currently support for two types of queries:

- Non-conflicting values
- Mandatory values.

Non-mandatory Properties

Resolved artifacts may either have no property with the key specified, or have the property with the key specified and the exact value specified (i.e. the artifact is resolved if it has a property with a non-conflicting value).

Non-mandatory properties are identified by a simple `key=value` parameter.

For example:

Current Artifact Property	Matrix Parameter	Resolution Result
<code>color=black</code>	<code>color=black</code>	OK (200)
None or <code>height=50</code>	<code>color=black</code>	OK (200)
<code>color=red</code>	<code>color=black</code>	NOT_FOUND (404)

Mandatory Properties

Resolved artifacts must have a property with the key specified and the exact value specified.

Mandatory properties are identified with a plus sign (+) after the property key: `key+=value`.

For example:

Current Artifact Property	Matrix Parameter	Resolution Result
color=black	color+=black	OK (200)
None or height=50	color+=black	NOT_FOUND (404)
color=red	color+=black	NOT_FOUND (404)

Multiple properties in queries

Multiple key-value matrix parameters are additive, forming an AND query between each key-value subsection.

Multi-valued Properties Support

All matrix parameters can support multiple values by separating values with a comma (.). For example:

```
colors=red,gold,green
```

Repository Layouts

Overview

Together with the growing number of choices for build-tools and frameworks there are also many ways in which modules can be stored within a repository.

Initially, Artifactory supported the Maven layout conventions for dealing with modules (and relying on Maven-specific metadata). However, your build tool should be able to "talk" to the repository "naturally", so if you are using Ivy or Gradle, there is no need to configure them to use the Maven conventions in order to "fit in". Moreover, combining and chaining repositories that use different layouts should work out-of-the-box.

This is where the Repository Layouts Add-on comes into play!

The Freedom of Custom Layouts

With the Repository Layouts Add-on, Artifactory allows you to take full control over the layout used by each repository and uses layout definitions to identify module artifacts and descriptors. By using repository layouts, Artifactory offers these smart module management facilities for any build technology:

- Automatic snapshot/integration versions cleanup
- Deleting old versions
- Conversions between remote and local layouts
- Conversions between 2 local layouts when moving or copying
- Resolution conversions from a virtual repository to its underlying repositories (where the virtual repository has its own layout defined)

Page Contents

- [Overview](#)
- [The Freedom of Custom Layouts](#)
- [Bundled Layouts](#)
- [Modules and Path Patterns used by Repository Layouts](#)
 - [Module Fields](#)
 - [Using Module Fields to Define Path Patterns](#)
 - [Path Pattern Tokens](#)
 - [Artifact Path Patterns](#)
 - [Descriptor Path Patterns](#)
- [Configuration](#)
 - [Layout Configuration](#)
 - [Testing Layouts](#)

- Path Patterns
- Regular Expressions for File and Folder Integration Revision
- Repository Configuration
 - Local Repository Configuration
 - Remote Repository Configuration
 - Virtual Repository Configuration

Bundled Layouts

Artifactory comes out-of-the-box with a number of default, predefined layouts requiring no additional configuration:

- **Maven 2/3**
- **Ivy** (default layout)
- **Gradle** (Wharf cache default layout)
- **Maven 1**

Support for repository layouts in Artifactory OSS

Layout configuration for conversion and resolution is available only to Artifactory Power Pack users. Users of the OSS version can only [Configure their Repositories](#) to use the default repository layouts bundled with Artifactory.

The OSS version only supports the automatic snapshot/integration version cleanup and deleting old version features.

Modules and Path Patterns used by Repository Layouts

Module Fields

To support smart module management, Artifactory must construct module information for stored files. Artifactory constructs this information based on path pattern information defined as part of the Repository Layout configuration (detailed below).

A module is comprised of various sub-elements or fields, which are typically expressed in the path of a stored artifact.

The module-sub elements recognized by Artifactory are listed below. At a minimum, there are three mandatory fields required for module identification:

- Organization
- Module
- Base Revision.

Field	Description	Example	Mandatory
Organization	A sequence of literals that identifies the artifact's organization	"org.slf4j"	✓
Module	A sequence of literals that identifies the artifact's module	"slf4j-api"	✓
Base Revision	A sequence of literals that identifies the base revision part of the artifact version, excluding any integration information	"1.5.10", or in case of an integration revision "1.2-SNAPSHOT" the base revision is "1.2"	✓
Folder Integration Revision	A sequence of literals that identifies the integration revision part used in folder names in the artifact's path, excluding the base revision	in case of an integration revision "1.2-SNAPSHOT" the folder integration revision is "SNAPSHOT"	✗
File Integration Revision	A sequence of literals that identifies the integration revision part in the artifact's file name, excluding the base revision	in case of an integration revision "1.2-20110202.144533-3" the file integration revision is "20110202.144533-3"	✗
Classifier	A sequence of literals that identifies the artifact's classifier	"sources"	✗
Extension	A sequence of literals that identifies the artifact's extension	"zip"	✗
Type	A sequence of literals that identifies the artifact's type. Typically used when the artifact's extension cannot be reused as the artifact's type	"java-source"	✗

Using Module Fields to Define Path Patterns

A path pattern is used in the configuration of a Repository Layout.

The pattern is similar to that of the Ivy pattern and is used to define a convention for artifact resolution and publication paths.

Artifactory uses path patterns to construct module information for stored files. This module information is then used to facilitate all the features mentioned above (version cleanup, cross-repo path conversions, etc.).

Path Pattern Tokens

A path pattern is constructed of tokens (explained below), path separators ('/'), optional parentheses ('(' and ')') and literals ('.', '-', etc.). Tokens are modeled after module fields, as presented above.

Path patterns can be defined for every artifact in the repository or you can define a separate path patterns for descriptor-type artifacts (such as, a .pom or an ivy.xml file).

The following tokens are available:

<code>[org]</code>	Represents the Organization field where the levels are separated by dots ('.'), a-la Ivy. For example: "org.slf4j"
<code>[orgPath]</code>	Represents the Organization field where the levels are separated by path separators ('/'), a la Maven. For example: "org/slf4j"
<code>[baseRev]</code>	Represents the Base Revision field
<code>[module]</code>	Represents the Module field
<code>[folderIntegRev]</code>	Represents the Folder Integration Revision field
<code>[fileIntegRev]</code>	Represents the File Integration Revision field
<code>[classifier]</code>	Represents the Classifier field
<code>[ext]</code>	Represents the Extension field
<code>[type]</code>	Represents the Type field
<code>[customTokenName<customTokenRegex>]</code>	A custom token. Can be used to create a new type of token when the provided defaults are insufficient. For example, <code>[myIntegRev<ITEG-(?:[0-9]+)>]</code> creates a new custom token named <code>myIntegRev</code> that matches the word <code>ITEG</code> followed by a dash and a minimum of one digit.

Custom tokens

When using custom tokens in the repository layout, make sure that the layout begins with `[orgPath]/[module]`. If the `[module]` token is missing in the layout, some REST API calls will not work.

Multiple Custom Tokens

Artifactory supports any number of custom tokens, but when provided with multiple custom tokens of the same key, Artifactory only takes into account the regular expression of the first occurrence while substituting the rest with a repetition expression (even if each occurrence has a different regular expression value).

For example:

```
[custom1<.+>]/[custom1<.*>]/[custom1<[0-9]+>]
```

Translates to:

```
<custom1>.+/\1/\1
```

Optional parts

To specify tokens or a sequence of tokens and literals as optional in the path pattern, surround the sequence with the optional parentheses '(' and ')' literals.

For example, the pattern "[module](-[classifier])" matches both "bobs-tools-sources" and "bobs-tools", and the pattern "[baseRev](-[fileItegRev])" matches both "1.2-SNAPSHOT" and "1.2".

Artifact Path Patterns

An artifact path pattern represents the typical structure that all module artifacts are expected to be stored in.

For example,

- To represent a normal Maven artifact path: "org/eclipse/jetty/jetty-ajp/7.0.2.v20100331/jetty-ajp-7.0.2.v20100331.jar"

Use the artifact path pattern:

```
[orgPath]/[module]/[baseRev](-[folderItegRev])/[module]-[baseRev](-[fileItegRev])(-[classifier]).[ext]
```

- To represent a default Ivy artifact path: "org.eclipse.jetty/jetty-ajp/7.0.2.v20100331/jars/jetty-ajp-7.0.2.v20100331.jar"

Use the artifact path pattern:

```
[org]/[module]/[baseRev](-[folderItegRev])/[type]s/[module]-[baseRev](-[fileItegRev])(-[classifier]).[ext]
```

Descriptor Path Patterns

A descriptor path pattern is used to recognize descriptor files (like .pom or ivy.xml files).

Using a specific descriptor path pattern is optional. When not used, Artifactory constructs module information for descriptor files using the artifact path pattern.

Even though descriptor paths patterns are optional, usage of them is **highly recommended** in cases of distinctive descriptors, such as Ivy ivy-1.0.xml and Maven bobs-tools-1.0.pom.

For example,

- To represent a normal Maven descriptor path: "org/eclipse/jetty/jetty-ajp/7.0.2.v20100331/jetty-ajp-7.0.2.v20100331.pom"

Use the descriptor path pattern:

```
[orgPath]/[module]/[baseRev](-[folderItegRev])/[module]-[baseRev](-[fileItegRev])(-[classifier]).pom
```

- To represent a default Gradle descriptor path: "org.eclipse.jetty/jetty-ajp/ivy-7.0.2.v20100331.xml"

Use the descriptor path pattern:

```
[org]/[module]/ivy-[baseRev](-[fileItegRev]).xml
```

Configuration

Repository layouts are configured on the global level of your Artifactory instance, so that any layout can be shared and reused across any number of repositories.

Layout Configuration

Layout configuration is available to administrator users in the **Admin** module under **Repositories | Layouts**.

11 Repository Layouts

Filter by Name

Page 1 of 1

Name	Artifact Path Pattern
bower-default	[orgPath]/[module]/[module]-[baseRev](-[fileItegRev]).[ext]
gradle-default	[org]/[module]/[baseRev](-[folderItegRev])/[module]-[baseRev](-[fileItegRev])(-[classifier]).[ext]
ivy-default	[org]/[module]/[baseRev](-[folderItegRev])/[type]s/[module](-[classifier])-[baseRev](-[fileItegRev]).[ext]
maven-1-default	[org]/[type]s/[module]-[baseRev](-[fileItegRev])(-[classifier]).[ext]
maven-2-default	[orgPath]/[module]/[baseRev](-[folderItegRev])/[module]-[baseRev](-[fileItegRev])(-[classifier]).[ext]
npm-default	[orgPath]/[module]/[module]-[baseRev](-[fileItegRev]).tgz
nuget-default	[orgPath]/[module]/[module].[baseRev](-[fileItegRev]).nupkg
sbt-default	[org]/[module]/(scala_[scalaVersion<, +>])/(sbt_[sbtVersion<, +>])/[baseRev]/[type]s/[module](-[classifier]).[ext]
simple-default	[orgPath]/[module]/[module]-[baseRev].[ext]
test	[orgPath]/[module]/[baseRev](-[folderItegRev])/[module]-[baseRev](-[fileItegRev])(-[classifier]).[ext]
vcs-default	[orgPath]/[module]/[refs<tags branches>]/[baseRev]/[module]-[baseRev](-[fileItegRev])(-[classifier]).[ext]

Duplicate

Additional layouts can be created from scratch by clicking "New" or by duplicating an existing layout.

New Repository Layout

Repository Layout Settings

Layout Name *

Artifact Path Pattern *

Distinctive Descriptor Path Pattern

Folder Integration Revision RegExp *

File Integration Revision RegExp *

Test Artifact Path Resolution

Test Path

Regular Expression View

Testing Layouts

Once you have finished configuring your layout, you can test it on an artifact path, and see how Artifactory would build module information from the path, using the layout definitions.

Path Patterns

These are used to define the artifact path pattern and the descriptor path pattern (optional), as explained above.

Use patterns in the directory part of the path

To achieve best path matching results, it is highly recommended that artifact and descriptor patterns also contain the mandatory tokens ([org] or [orgPath], [module] and [baseRev]) within the directory structure itself.

For example, Gradle's artifact path pattern:

```
[org]/[module]/[baseRev](-[folderItegRev])/[module]-[baseRev](-[fileItegRev])(-[classifier]).[ext]
```

Regular Expressions for File and Folder Integration Revision

These fields should contain regular expressions that exactly match and describe the integration revision (excluding the base revision) formats as they are expected in the artifact's file name and path-structure folder name.

Avoid using capturing group syntax in regexp

Regular expressions entered in these fields are wrapped and treated as a single capturing group.

Refrain from introducing any capturing groups within the expressions. Failure to do so may result in unexpected behavior and compromise the accuracy of the matcher.

Folder integration revision regular expression examples:

- Maven's folder integration revision is simply the constant `-SNAPSHOT` appended to the base revision ("1.2-SNAPSHOT"), so the regular expression is

```
SNAPSHOT
```

- Ivy's default folder integration revision is usually equal to the file integration revision and is normally a 14 digit timestamp ("5.1-20101202161531"), so the regular expression can be

```
\d{14}
```

File integration revision regular expression examples:

- Maven's file integration revision can be either the `-SNAPSHOT` constant ("1.2-SNAPSHOT") or a timestamp, where the date and time are separated by a dot ('.'), with an addition of a dash ('-') and a build-number ("2.3-20110108.100922-2"), so the regular expression should be able to fit them both

```
SNAPSHOT | ( ? : ( ? : \d { 8 } . \d { 6 } ) - ( ? : \d + ) )
```

- Ivy's default file integration revision is normally a 14 digit timestamp ("5.1-20101202161531") and usually equal to the folder integration revision, so the regular expression may be the same as suggested in the file's example

```
\d{14}
```

Repository Configuration

Before custom layouts

Repositories created prior to the introduction of custom repository layouts are automatically configured with the default Maven 2 layout.


Local Repository Configuration

Layouts are mandatory for local repositories, since they define the structure with which artifact are stored.

When you create a new repository, Artifactory will recommend the best layout according to the **Package Type** selected for the repository.

Basic > Advanced > Replications

Package Type *

 Npm

Repository Key *

npm-local

General

Repository Layout

Select Repository Layout...

- npm-default
- bower-default
- vcs-default
- sbt-default
- simple-default

Internal Description

Excludes Pattern

Remote Repository Configuration

Layouts are mandatory only for the remote repository cache configuration, however, you can also specify the layout of the remote repository itself.


If the remote repository itself uses a different layout than the one chosen for the cache, all requests to the remote target are translated from the path of the cache layout to the path of the remote layout.

For example, the remote repository <http://download.java.net/maven/1> stores its artifacts according to the Maven 1 convention. You can configure the cache of this repository to use the Maven 2 layout, but set the **Remote Layout Mapping** to Maven 1. This way, the repository cache handles Maven 2 requests and artifact storage, while outgoing requests to the remote repository are translated to the Maven 1 convention.

Edit java.net.m1 Repository

Basic Advanced Replications

Package Type *


Maven

Repository Key * URL *

java.net.m1 http://download.java.net/maven/1

General

Repository Layout Remote Layout Mapping

maven-2-default maven-1-default ✕

Public Description Internal Description

java.net Maven1 Format

Includes Pattern Exclude Pattern

**/*

Offline

Virtual Repository Configuration

You can also configure a layout for a virtual repository.

When configured, all resolution requests can be made according to the virtual repository layout. When trying to resolve requests to the virtual repository Artifactory attempts to translate the request path to the layout of each nested repository, according to the module information constructed from the virtual request.

In the following cases, the request path is not translated, and requests pass through to nested repositories with the original specified path:

- Module information cannot be constructed
- The virtual module information cannot be mapped to a nested repository (e.g., fields are missing on one of the sides)
- The virtual repository or the nested repository are not configured with a layout

Repository Replication

Overview

Through the Replication Add-on in Artifactory Pro, Artifactory allows replication of repositories between two Artifactory instances to support development by different teams distributed over distant geographical sites. The benefits of replication are:

- Ensuring developers all work with the same version of remote artifacts
- Ensuring build artifacts are shared efficiently between the different development teams
- Overcome connectivity issues such as network latency and stability when accessing remote artifacts
- Accessing specific versions of remote artifacts

Artifactory versions for replication

We strongly recommend that replication is only performed between servers running the

Page Contents

- Overview
 - Push Replication
 - Advantages
 - Multi-push Replication
 - Pull Replication
 - Advantages
- Scheduling and Configuring Replication
 - Using the UI

same version of Artifactory Pro.

Two main methods of replication are supported:

- **Push replication**
Both scheduled and event-based push replication are supported, and multi-push replication is available with an Enterprise license
- **Pull replication**
Both scheduled and event-based pull replication are supported; event-based pull requires an Enterprise license.

- Configuring Push Replication
- Adding a push replication target
- Configuring Pull Replication
- Replicating with REST API
- Replication Properties
- Watch the Screencast

Push Replication

Push replication is used to synchronize [Local Repositories](#), and is implemented by the Artifactory server on the near end invoking a synchronization of artifacts to the far end.

There are two ways to invoke push replication:

- **Scheduled push:** Pushes are scheduled asynchronously at regular intervals
- **Event-based push:** Pushes occur in nearly in real-time since each create, copy, move or delete of an artifact is immediately propagated to the far end.

Advantages

- It is fast because it is asynchronous.
- It minimizes the time that repositories are not synchronized.
- It reduces traffic on the master node in case of a replication chain ("Server A" replicates to "Server B", "Server B" then replicates to "Server C" etc.).

Avoid Replication Loops ("Cyclic Replication")

A replication loop occurs ("Cyclic" or "Bi-directional" replication) occurs when two instances of Artifactory running on different servers are replicating content from one to the other concurrently.

For example, "Server A" is configured to replicate its repositories to "Server B", while at the same time, "Server B" is configured to replicate its repositories to "Server A".

Or "Server A" replicates to "Server B" which replicates to "Server C" which replicates back to "Server A".

We strongly recommend avoiding cyclic replication since this can have disastrous effects on your system causing loss of data, or conversely, exponential growth of disk-space usage.



Replication loop to be strictly avoided

When to Use Push Replication

Event-based push replication is recommended when it is important for the repository at the far end to be updated in near-real-time for any change (create, copy, move or delete of an artifact) in the repository at the near end.

Regular scheduled replications run on top of event-based replication to guarantee full copy consistency even in cases of server downtime and network partitions.

Multi-push Replication

Requires an Enterprise license

With an Enterprise license, Artifactory supports multi-push replication allowing you to replicate a local repository from a single source to multiple enterprise target sites simultaneously.

Pull Replication

This provides a convenient way to proactively populate a remote cache, and is very useful when waiting for new artifacts to arrive on demand (when first requested) is not desirable due to network latency.

There are two ways to invoke a pull replication:

- **Scheduled pull:** Pull replication is invoked by a remote repository, and runs asynchronously according to a defined schedule to synchronize repositories (local, remote or virtual) at regular intervals.
- **Event-based pull:**

Requires an Enterprise license

Pulls occur nearly in real-time since each create, copy, move or delete of an artifact is immediately propagated to the far end. As soon as a file is uploaded it is replicated and immediately available to the target (pulling) instance without even having to wait for the file upload to be completed at the source

Advantages

- Many target servers can pull from the same source server efficiently implementing a one-to-many replication.
- It is safer since each package only has one "hop".
- It reduces traffic on target servers since they do not have to pass on artifacts in a replication chain.

When and when not to Use Pull Replication

Pull replication is recommended in the following cases:

- When you need to replicate a repository to many targets.
- When your source repository is located behind a proxy that prevents push replication (e.g. replicating a repository hosted on Artifactory SaaS to a local repository at your site)

Pull replication cannot be used to replicate a remote resource that is not an Artifactory repository. Artifacts from third party repositories can only be cached on-demand in the normal cache and proxy behavior of a [remote repository](#).

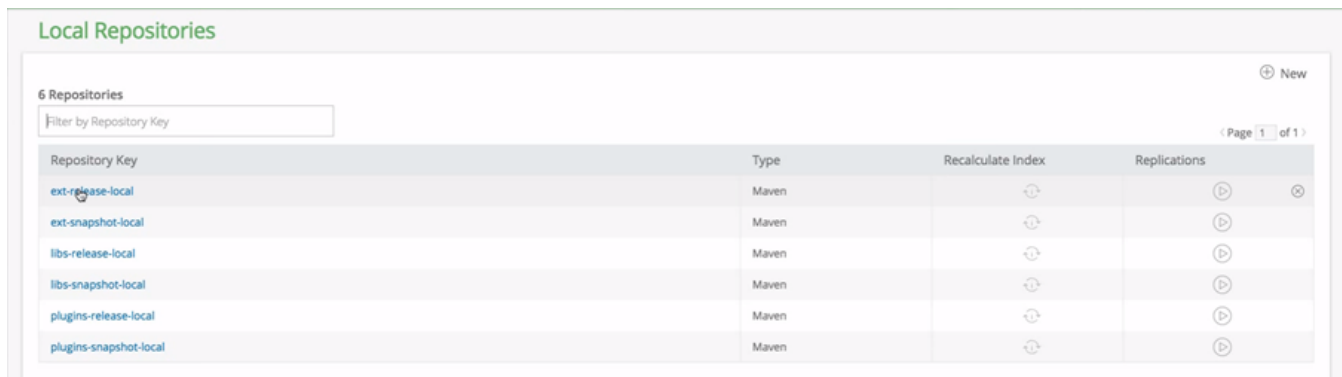
Scheduling and Configuring Replication

Using the UI

Replication is configured via the user interface as a scheduled task. Local repositories can be configured for push replication, and remote repositories can be configured for pull replication.

All replication messages are logged in the main [Artifactory log file](#) (*artifactory.log*).

The **Replications** column in your list of local repositories indicates if replication is configured for each repository in the list. If replication is indeed configured for a repository, you can click the icon in the list to invoke it.



The screenshot shows the 'Local Repositories' section of the Artifactory UI. It features a search filter 'Filter by Repository Key' and a 'New' button. Below is a table with the following data:

Repository Key	Type	Recalculate Index	Replications
ext-release-local	Maven		
ext-snapshot-local	Maven		
libs-release-local	Maven		
libs-snapshot-local	Maven		
plugins-release-local	Maven		
plugins-snapshot-local	Maven		

Configuring Push Replication

A push replication task for a Local Repository is configured in the **Replication** tab of the [Edit Local Repository](#) dialog.

First, in the **Cron Expression** field define the replication task schedule using a valid [cron](#) expression.

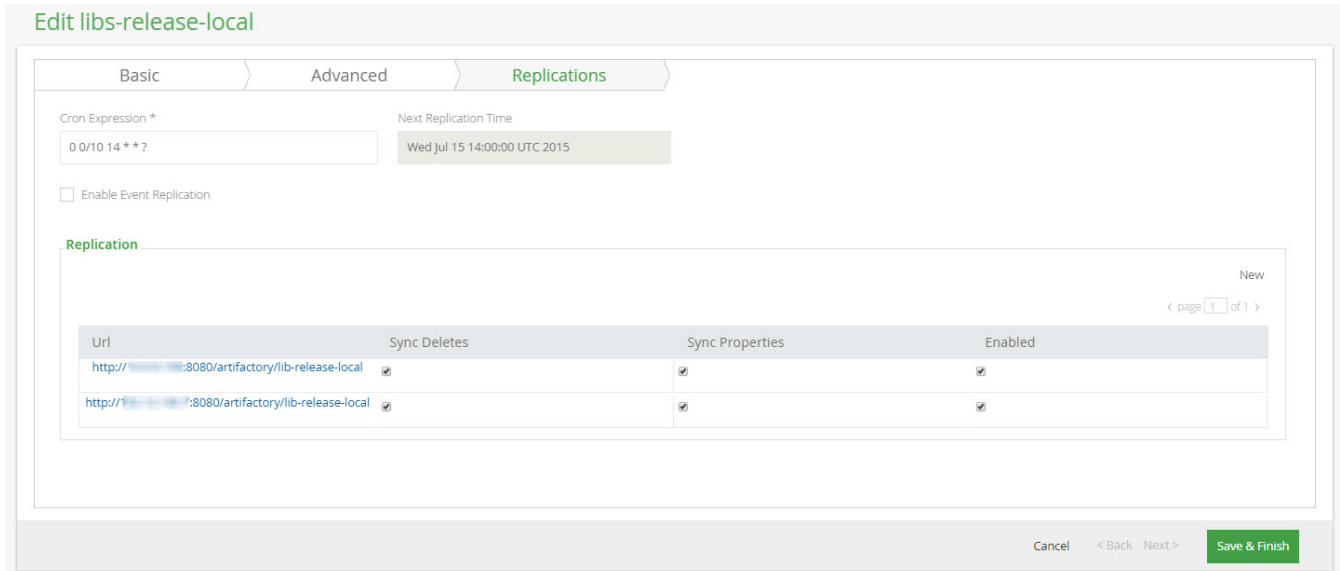
The **Next Replication Time** will indicate update accordingly.

Cron Expression VS Event Base Replication

Replication of this repository to all of its targets occurs simultaneously according to the **Cron Expression** you define.

The event base replication will attempt to replicate **only** the artifacts affected by the event while the Cron Expression will trigger a sync of all artifacts in repository. This difference is important since in case one of the event sync has failed the next time the Cron Expression will trigger a sync all changed will be synced.

Once you have configured the replication properties for each of your replication targets, the **Replication** tab for your repository displays them.



Field Name	Description
Push to	The replication targets you have defined
Enabled	When set, enables replication of this repository to the target specified in Push to
Enable Event Replication	When set, each event will trigger replication of the artifacts changed in this event. This can be any type of event on artifact, e.g. add, deleted or property change.

Number of replication targets

If you do not have an Enterprise license, you may only define **one** replication target. With an Enterprise license, Artifactory supports multi-push replication and you may define as many targets as you need.

Adding a push replication target

To add a target site for this replication, click **Add** to display the **Replication Properties** dialog, and fill in the details as follows.

New Replication ✕

Enabled

URL *

Username * Password

Network Proxy Reference Socket Timeout *

Sync Deleted Artifacts Sync Artifact Properties

Sync Artifact Statistics

Path Prefix ?

Field Name	Description
Enable Active Replication of this Repository	When set, this replication will be enabled when saved
URL	The URL of the target local repository on a remote Artifactory server.
Username	The HTTP authentication username.
Password	The HTTP authentication password.
Proxy	A proxy configuration to use when communicating with the remote instance.

Socket Timeout	The network timeout in milliseconds to use for remote operations.
Sync Deleted Artifacts	When set, items that were deleted locally should also be deleted remotely (also applies to properties metadata).
Sync Artifact Properties	When set, the task also synchronizes the properties of replicated artifacts.
Sync Artifact Statistics	When set, the task also synchronizes artifact download statistics. Set to avoid inadvertent cleanup at the target instance when setting up replication for disaster recovery.
Path Prefix (optional)	Only artifacts that located in path that matches the subpath within the repository will be replicated.

Configuring Pull Replication

A pull replication task for a Remote Repository is configured in the **Replication** tab of the [Edit Remote Repository](#) dialog.

First, in the **Cron Expression** field define the replication task schedule using a valid [cron](#) expression.

The **Next Replication Time** will indicate update accordingly.

New Remote Repository

Basic
Advanced
Replications

Enable Active Replication of This Repository

Cron Expression * ?

Next Replication Time

Enable Event Replication ?

Sync Deleted Artifacts ?

Sync Artifact Properties ?

Path Prefix ?

Field Name	Description
------------	-------------

Enable Active Replication of this Repository	When set, this replication will be enabled when saved
URL	The URL of the target local repository on a remote Artifactory server. For some package types, you need to prefix the repository key in the URL with api/<pkg> . For a list of package types where this is required, see the note below .
Sync Deleted Artifacts	When set, items that were deleted locally should also be deleted remotely (also applies to properties metadata).
Sync Artifact Properties	When set, the task also synchronizes the properties of replicated artifacts.
Path Prefix (optional)	Only artifacts that located in path that matches the subpath within the remote repository will be replicated.
Enable Event Replication	When set, each event will trigger replication of the artifacts changed in this event. This can be any type of event on artifact, e.g. added, deleted or property change.

Regarding credentials of the remote repository configuration

The remote repository's file listing for replication is retrieved using the repository's credentials defined under the repository's [Advanced](#) configuration section.

The remote files retrieved depend on the effective permissions of the configured user on the remote repository (on the other Artifactory instance).

* Check for which package formats you need to prefix the repository path with **api/<pkg>**

For some packaging formats, when using the corresponding client to access a repository through Artifactory, the repository key in the URL needs to be prefixed with **api/<pkg>** in the path. For example, in the case of [Npm](#) repositories, the repository key should be prefixed with `api/npm`.

Nevertheless, there are exceptions to this rule. For example, when replicating Maven repositories, you do **not** need to add a prefix the remote repository path.

The considerations of whether to prefix the repository key with **api/<pkg>** or not are the same as those when configuring smart remote repositories. For a detailed list of package formats that should be prefixed with **api/<pkg>**, please refer to [Configuration under Smart Remote Repositories](#).

Replicating with REST API

Both Push and Pull Replication are supported by Artifactory's REST API. For details please refer to the following:

- [Get Repository Replication Configuration](#)
- [Set Repository Replication Configuration](#)
- [Update Repository Replication Configuration](#)
- [Delete Repository Replication Configuration](#)
- [Scheduled Replication Status](#)
- [Pull/Push Replication](#)

Replication Properties

Once replication has been invoked, Artifactory annotates the source repository being replicated and annotates it with properties that indicate the status of the replication. These can be viewed, along with other properties that may annotate the repository, in the **Properties** tab of the **T** [ree Browser](#).

For single push replication operations, the following properties are created/updated:

Key	Value
artifactory.replication.<source_repo_key>.started	Indicates when the replication started
artifactory.replication.<source_repo_key>.status	Indicates the status of the replication operation once complete. It can take the following values: ok : The replication succeeded failure : The replication failed. You should check the log files for errors
artifactory.replication.<source_repo_key>.finished	Indicates when the replication finished

Property	Value(s)
artifactory.replication.libs-snapshot-local.started	1437483921588
artifactory.replication.libs-snapshot-local.finished	1437484260530
artifactory.replication.libs-snapshot-local.result	ok

For multi-push replication operations (available to Enterprise customers only), the following properties are created/updated:

Key	Value
artifactory.replication.<source_repo_key>_<target_repo_URL>.started	Indicates when the replication started
artifactory.replication.<source_repo_key>_<target_repo_URL>.status	Indicates the status of the replication operation once complete. It can take the following values: ok : The replication succeeded failure : The replication failed. You should check the log files for errors
artifactory.replication.<source_repo_key>_<target_repo_URL>.finished	Indicates when the replication finished

Watch the Screencast

To see replication in action, watch the short screencast below.

S3 Object Storage

Overview

Artifactory fully supports S3 object storage for distributed file systems so your Artifactory filestore can reside on the cloud. This presents several benefits:

1. **Unlimited scalability**

Since your files are now stored on the cloud, this means that your Artifactory filestore is scalable and effectively unlimited (to the extent offered by your storage provider). You may freely continue to upload files without having to install or maintain any file storage devices. You can even upload files larger than 5 GB using multi-part upload.

2. Security

Enjoy the same security and authentication mechanisms provided by your S3 provider.

3. Disaster recovery

Since your files are replicated and stored with redundancy, this offers the capability for disaster recovery.

4. Support any S3 compatible distributed file system

Artifactory's support is based on the S3 protocol. Any provider that uses S3, such as Ceph, Swift (through the S3 API) and others, will also be supported by Artifactory. With support for AWS S3 version 4, you can sign AWS requests using [Signature Version 4](#).

Support for S3 object storage is included with an [Artifactory Enterprise license](#).

Backup your system. Your current filestore will be deleted.

Setting up Artifactory to use S3 will delete all files in your current filestore.

If you already have a running installation of Artifactory, then before you setup Artifactory to use S3 and migrate your filestore to the cloud, we strongly recommend that you do a [complete system backup](#).

 Requires an **Enterprise license**

Page Contents

- Overview
- Setting up Artifactory to Use S3
 - Setting Your License
 - Configuring Artifactory to Use S3
 - Migrating Your Filestore from local/mounted storage to S3
 - Automatic Filestore Migration (Recommended)
 - Manual Filestore Migration

Setting up Artifactory to Use S3

First time installation or upgrade

If you are moving your filestore to S3 in the context of upgrading Artifactory, or a first time installation, we recommend that you first do a standard installation of Artifactory using the default settings, or a standard upgrade using your current settings.

In order to move your Artifactory filestore to the cloud, you need to execute the following steps:

- Shut down Artifactory.
- Set your enterprise license
- Configure Artifactory to use your S3 object storage provider
- Migrate your files to the cloud manually or automatically
- Start up Artifactory

Setting Your License

To use an S3 object store, your Artifactory installation needs to be activated with an [Enterprise license](#).

Configuring Artifactory to Use S3

From version 4.6, Artifactory's filestore is configured through the `binarystore.xml` file. For details, please refer to [Configuring the Filestore](#).

Migrating Your Filestore from local/mounted storage to S3

- For an Artifactory **HA cluster** running version 5.0 and above, to migrate your filestore to an S3 provider, please refer to [Migrating Data from NFS](#) Wiki page.

Standalone installations: there are two ways to migrate your filestore over to your S3 provider.

- [Automatically](#) (recommended)
- [Manually](#)

Automatic Filestore Migration (Recommended)

To make sure your filestore migration completes successfully without corrupting files, we recommend configuring Artifactory to do this migration for you automatically:

To do so, you need to create the following links in `ARTIFACTORY_HOME/data/eventual/` (create it if the `eventual` folder does not exist - it is created automatically when the eventual binary provider is applied via an Artifactory restart with an updated `binarystore.xml`):

- A link with the name `_add` that points to the `ARTIFACTORY_HOME/data/filestore` directory
- A link with the name `_pre` that points to the `ARTIFACTORY_HOME/data/_pre` directory

With this setting, as soon as Artifactory starts up, it will automatically move your complete filestore over to your S3 provider.

Your current filestore will be deleted

The process of moving your filestore to your S3 provider will delete your current filestore. We strongly recommend you do a [complete system backup](#) before doing this migration.

Once the migration is complete, you may delete the `_pre` link and the `ARTIFACTORY_HOME/data/_pre` directory

Manual Filestore Migration

To migrate your filestore manually, you need to execute the following steps:

- Stop Artifactory
- Copy the `ARTIFACTORY_HOME/data/filestore` directory to your S3 object storage to the bucket name and path specified when you [configured Artifactory to use S3](#).
- Start Artifactory

SAML SSO Integration

SAML (Security Assertion Markup Language)

SAML is an XML standard that allows you to exchange user authentication and authorization information between web domains.

Artifactory offers a SAML-based Single Sign-On service allowing federated Artifactory partners (identity providers) full control over the authorization process.

Using SAML, Artifactory acts as service provider which receives users' authentication information from

external identity providers.

In this case, Artifactory is no longer responsible for authentication of the user although it still has to redirect the login request to the identity provider and verify the integrity of the identity provider's response.

Page Contents

- SAML (Security Assertion Markup Language)
- Artifactory's SAML configuration
- Understanding Artifactory's SAML-based SSO Login Process
- Understanding the Artifactory's SAML-based SSO Logout Process
- Artifactory Profiles and Bindings
 - After SAML Setup
 - Login Failure
- Using API Key with SAML Users

Artifactory's SAML configuration

SAML SSO integration is configured in the **Admin** module under **Security | SAML SSO**.

SAML SSO Configuration

SAML SSO Settings

Enable SAML Integration

SAML Login URL * 

https://jfrog.

SAML Logout URL * 

https://jfrog.

SAML Service Provider Name * 

SAML Certificate 

Auto Associate Groups 

Group Attribute ?

Groups

Email Attribute ?

email

Auto Create Artifactory Users ?

Allow Created Users Access To Profile Page ?

Auto Redirect Login Link To SAML Login ?

Enable SAML Integration	When checked, SAML integration is enabled and users may be authenticated via a SAML server.
SAML Login URL	The SAML login URL.
SAML Logout URL	The SAML logout URL.
SAML Service Provider Name	The SAML service provider name. This should be a URI that is also known as the entityID, providerID, or entity identity. For more details, see section 8.3.6 of the SAML v2 specification .
Auto Associate Groups	<p>When set, in addition to the groups the user is already associated with, he will also be associated with the groups returned in the SAML login response.</p> <p>Note that the user's association with the returned groups is not persistent. It is only valid for the current login session in the browser (i.e. this will not work for logins using the SAML user id and API Key).</p> <p>Also, the association will not be reflected in the UIs Groups settings page. Instead, you can see this by enabling this SAML logger in your <code>\$ARTIFACTORY_HOME/etc/logback.xml</code> file as follows:</p> <pre><logger name="org.artifactory.addon.sso.saml"> <level value="debug"/> </logger></pre>
Group Attribute	The group attribute in the SAML login XML response. Note that Artifactory will search for a case-sensitive match to an existing group.
Email Attribute	If Auto Create Artifactory Users is enabled or an internal user exists, Artifactory will set the user's email to the value in this attribute that is returned by the SAML login XML response.
SAML Certificate	The X.509 certificate that contains the public key.

Auto Create Artifactory Users	When checked, for new users accessing Artifactory in for the first time via SAML, Artifactory will create a user that will persist in the data base.
Allow Created Users Access To Profile Page	When checked, users created after authenticating using SAML, will be able to access their profile . This means they are able to generate their API Key and set their password for future use.
Auto Redirect Login Link to SAML Login	When checked, clicking on the login link will direct the users to the configured SAML login URL.

To use SAML-based SSO in Artifactory:

1. Login to Artifactory with administrator privileges.
2. In the **Admin** module, go to **Security | SAML SSO**.
3. Enable the SAML integration by checking the **Enable SAML Integration** checkbox.
4. Enable or disable "Auto Create Artifactory users" (Using SAML login). If enabled, new users will persist in the database.
5. Enable or disable "Allow Users Access to Profile Page". If enabled users will be able to [access their profile](#) without having to provide a password.
6. Provide the **SAML Login URL** and **SAML Logout URL**

SAML Logout URL

In order to simultaneously logout from your SAML provider and Artifactory, you need to correctly set your provider's logout URL **SAML Logout URL** field. Setting this incorrectly will keep your users logged in with the SAML provider even after logging out from Artifactory.

7. Provide the service provider name (Artifactory name in SAML federation)
8. Provide the X.509 certificate that contains the public key. The public key can use either the DSA or RSA algorithms. Artifactory uses this key to verify SAML response origin and integrity. Make sure to match the embedded public key in the X.509 certificate with the private key used to sign the SAML response.

Custom URL base

For your SAML SSO settings to work, make sure you have your [Custom URL Base](#) configured.

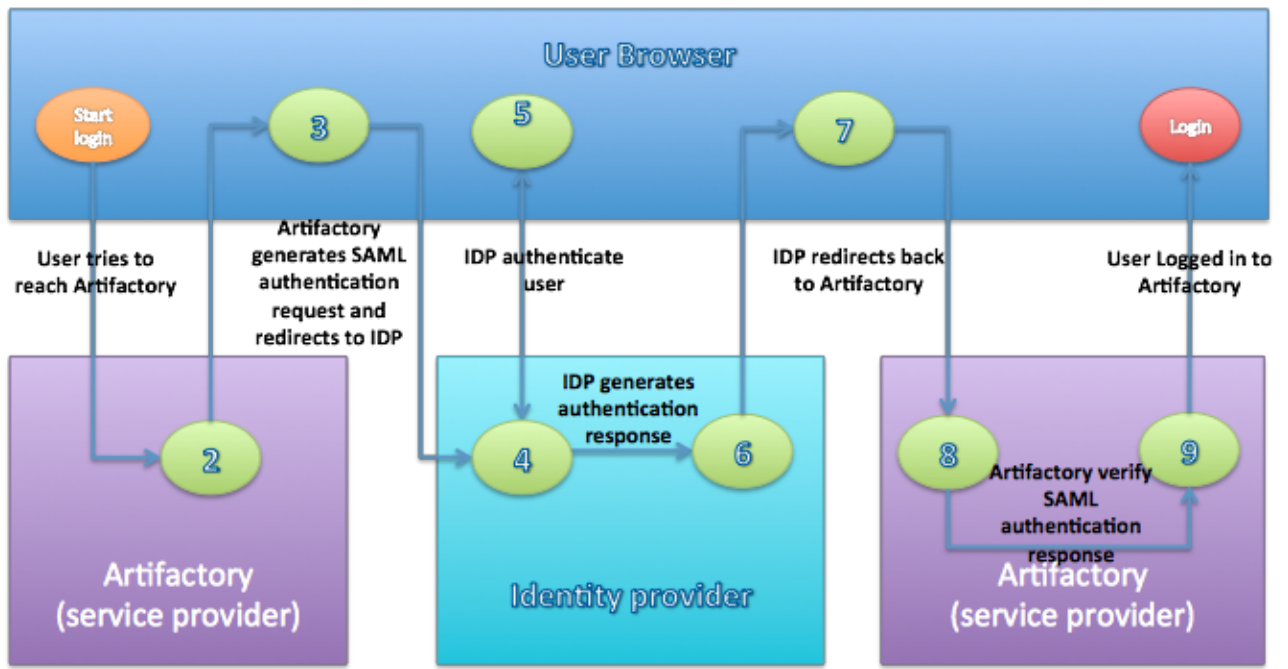
Signed and encrypted Assertions

1. Please make sure your SAML IdP (Identity Provider) provides a signed login Assertion - this is mandatory for the Assertion verification by Artifactory.
2. Encrypted Assertion is currently unsupported by Artifactory.
3. Signed Logout is also currently unsupported by Artifactory.

Understanding Artifactory's SAML-based SSO Login Process

1. The user attempts to reach a hosted Artifactory, Home Page.
2. Artifactory generates a SAML authentication request.
3. The SAML request is encoded and embedded into the identity provider URL.
4. Artifactory sends a redirect to the user's browser. The redirect URL includes the encoded SAML authentication request that should be submitted to the identity provider.
5. The identity provider decodes the SAML message and authenticates the user. The authentication process can proceed by asking for valid login credentials or by checking for valid session cookies.
6. The identity provider generates a SAML response that contains the authenticated user's username. In accordance with the SAML 2.0 specification, this response is digitally signed with the identity provider's private DSA/RSA keys.
7. The identity provider encodes the SAML response and returns that information to the user's browser. The identity provider redirects back to Artifactory with the signed response.
8. Artifactory's ACS verifies the SAML response using the partner's public key. If the response is successfully verified, the ACS redirects the user to the destination URL.
9. The user has been redirected to the destination URL and is logged in to Artifactory.

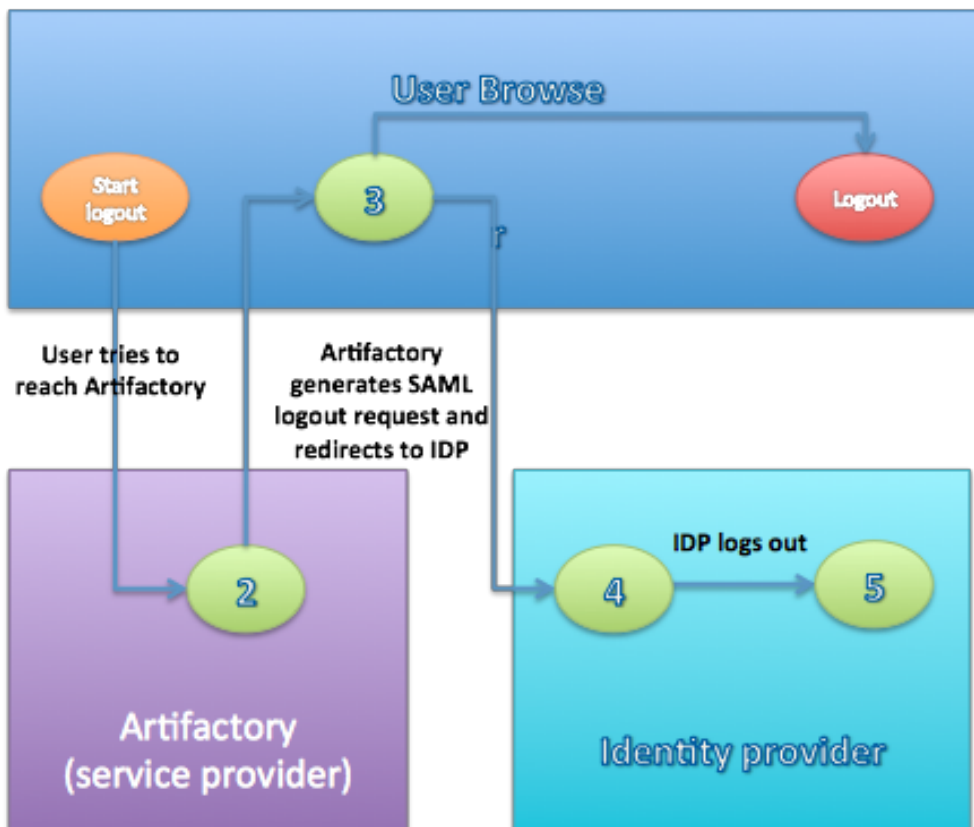
Figure (2) Artifactory's SAML-based SSO login process.



Understanding the Artifactory's SAML-based SSO Logout Process

1. The user attempts to reach a hosted Artifactory logout link.
2. Artifactory logs the client out and generates a SAML logout request.
3. Artifactory redirects to the identity provider with the encoded SAML logout request.
4. The identity provider decodes the SAML message and logs the user out.
5. The user is redirected to the configured URL in the identity provider.

Figure (3) Artifactory's SAML-based SSO logout process.



Artifactory Profiles and Bindings

Artifactory currently supports the Web Browser SSO and Single Logout Profiles.

The Web Browser SSO Profile uses HTTP redirect binding to send the AuthnRequest from the service provider to the identity provider, and HTTP POST to send the authentication response from the identity provider to the service provider.

Similar to the previous profile, the Single Logout Profile uses HTTP redirect binding to send the LogoutRequest from the service provider to the identity provider and HTTP POST to send the logout response from the identity provider to the service provider.

If your IDP supports uploading service provider metadata, you can use the following metadata XML:

Figure (4) Artifactory's service provider metadata XML.

```


Artifactory SP metadata XML

<ns2:EntityDescriptor xmlns="http://www.w3.org/2000/09/xmldsig#"
xmlns:ns2="urn:oasis:names:tc:SAML:2.0:metadata"
entityID="<SP_NAME_IN_FEDERATION>">
  <ns2:SPSSODescriptor WantAssertionsSigned="true"
AuthnRequestsSigned="false"
protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">

  <ns2:NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</n
s2:NameIDFormat>
  <ns2:AssertionConsumerService index="1"
Location="<ARTIFACTORY_URL>/webapp/saml/loginResponse"
Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" />
  </ns2:SPSSODescriptor>
</ns2:EntityDescriptor>
```

NOTE! that to use the service provider metadata:

Do not forget to update the following fields in the service provider metadata XML:

- entityID - Artifactory's ID in the federation. Must match [SAML Service Provider Name](#) in Artifactory's SAML configuration page.
- Location - Artifactory's home URL

After SAML Setup

Using SAML, Artifactory automatically redirects the request to IDP which Authenticates the user and after a successful login redirects back to Artifactory.

If "Anonymous User" is enabled, Artifactory doesn't have to authenticate the user therefore it doesn't redirect to the IDP. If the user still wants to sign in through SAML, they can do so by clicking the "SSO login" link in the login page.

Login Failure

In case of IDP failover or bad configuration, Artifactory allows you to bypass SAML login by using Artifactory login page:

`http://<ARTIFACTORY_URL>/webapp/#/login`

This URL can be used by internal users who need to log in directly to Artifactory.

Using API Key with SAML Users

While SAML provides access to Artifactory UI, it is also possible for SAML users to generate an [API key](#) that can be used instead of a password for basic authentication or in a dedicated [REST API header](#), this is very useful when working with different clients, e.g. docker, npm, maven, etc. or using Artifactory REST API.

In order to allow SAML users access to an API key you will need to make sure that the **"Auto Create Artifactory Users"** and **"Allow Created**

Users Access To Profile Page" check boxes are checked. This means that SAML users are also saved in Artifactory database and can access their [profile page](#) in order to generate, retrieve and revoke their API key.

Single Sign-on

Overview

The Single Sign-on (SSO) add-on allows you to reuse existing HTTP-based SSO infrastructures with Artifactory, such as the SSO modules offered by Apache HTTPd.

You can have Artifactory's authentication work with commonly available SSO solutions, such as native NTLM, Kerberos etc.

SSO works by letting Artifactory know what trusted information it should look for in the HTTP request, assuming this request has already been authenticated by the SSO infrastructure that sits in front of Artifactory.

Page Contents

- [Overview](#)
- [Usage](#)
- [Integrating Apache and Tomcat](#)
- [Setting Up a Reverse SSL Proxy for SSO](#)
 - [Components and Versions](#)
 - [Modifying Your Webserver Configuration File](#)
- [Using API Key with HTTP-SSO Users](#)

Usage

To access the Single Sign-On (SSO) add-on, in the **Admin** module, select **Security | HTTP SSO**.

To enable SSO you must alert Artifactory that it is running behind a secure HTTP server that forwards trusted requests to it.

Then you must tell Artifactory in which variable to look for trusted authentication information.

The default is to look for a REMOTE_USER header or the request variable, which is set by Apache's AJP and JK connectors.

You can choose to use any request attribute (as defined by the Servlet specification) by providing a different variable name.

Adding Your Own SSO Integration

You can write a simple servlet filter to integrate with custom security systems and set a request attribute on the request to be trusted by the SSO add-on.

Finally, you can instruct Artifactory to treat externally authenticated users as temporary users, so that Artifactory does not create them in its security database.

In this case, permissions for such users are based on the permissions given to auto-join groups.

HTTP SSO Configuration

HTTP SSO Settings

Artifactory is Proxied by a Secure HTTP Server [?](#)

Remote User Request Variable [?](#)

REMOTE_USER

Auto Create Artifactory Users [?](#)

Allow Users Access To Profile Page [?](#)

Field Name	Description
Artifactory is Proxied by a Secure HTTP Server	<p>When checked, Artifactory trusts incoming requests and reuses the remote user originally set on the request by the SSO of the HTTP server.</p> <p>This is extremely useful if you want to use existing enterprise SSO integrations, such as the powerful authentication schemes provided by Apache (mod_auth_ldap, mod_auth_ntlm, mod_auth_kerb, etc.).</p> <p>When Artifactory is deployed as a webapp on Tomcat behind Apache:</p> <ul style="list-style-type: none">• If using mod_proxy_ajp, make sure to set tomcatAuthentication="false" on the AJP connector.• If using mod_jk, make sure to use the "JkEnvVar REMOTE_USER" directive in Apache's configuration.
Remote User Request Variable	<p>The name of the HTTP request variable to use for extracting the user identity. Default is: REMOTE_USER.</p>
Auto Create Artifactory Users	<p>When not checked, authenticated users are not automatically created inside Artifactory. Instead, for every request from a SSO user, the user is temporarily associated with default groups (if such groups are defined) and the permissions for these groups apply.</p> <p>Without auto user creation, you must manually create the user inside Artifactory to manage user permissions not attached to its default groups.</p>
Allow Created Users Access To Profile Page	<p>When checked, users created after authenticating using HTTP SSO, will be able to access their profile. This means they are able to generate their API Key and set their password for future use.</p>

Custom URL base

For your HTTP SSO settings to work, make sure you have your [Custom URL Base](#) configured.

When Artifactory is deployed as a webapp on Tomcat behind Apache:

- If using **mod_proxy_ajp** - Make sure to set `tomcatAuthentication="false"` on the AJP connector.
- If using **mod_jk** - Make sure to use the `JkEnvVar REMOTE_USER` directive in Apache's configuration.
- If using **mod_proxy** (requires **mod_proxy_http**, **mod_headers** and **mod_rewrite** - There are two known working methods that forward the header:

```
RequestHeader set REMOTE_USER %{REMOTE_USER}e
```

or

```
RewriteEngine On
RewriteCond %{REMOTE_USER} (.+)
RewriteRule . - [E=RU:%1]
RequestHeader set REMOTE_USER %{RU}e
```

Setting Up a Reverse SSL Proxy for SSO

You may set up a reverse SSL proxy on your webserver in order to run Artifactory supporting SSO.

To do this, you need to have the right [components](#) installed, [modify your webserver configuration file](#), and then [configure Artifactory for SSO](#).

When correctly set up, you should be able to login to Artifactory with your Windows credentials and stay logged in between sessions.

Components and Versions

The instructions below have been tested to work with Kerberos/NTLM SSO working with Artifactory using the following components.

- [IBM Websphere 8.5.5](#) running on Windows 8 using the [IBM Websphere Java 7 JDK Package](#).
- Artifactory v3.3.0.1 or later must be installed on the Websphere instance. For details please refer to [Running Artifactory on IBM WebSphere](#).
- The [mod_auth_sspi](#) Apache module.

Modifying Your Webserver Configuration File

Once you have the right components and versions installed, you need to add the following lines to your `[HTTP_SERVER_HOME]/conf/httpd.conf` file:

httpd.conf file

```
<VirtualHost *:80>
  ServerName yourhostname
  DocumentRoot "C:/IBM/Installation
  Manager/eclipse/plugins/org.apache.ant_1.8.3.v20120321-1730"
  ProxyPreserveHost on
  ProxyPass /artifactory http://yourhostname:9080/artifactory
  ProxyPassReverse /artifactory http://yourhostname:9080/artifactory
</VirtualHost>

<Location /artifactory>
  AuthName "Artifactory Realm"
  AuthType SSPI
  SSPIAuth On
  SSPIAuthoritative On
  require valid-user
  RewriteEngine On
  RewriteCond %{REMOTE_USER} (.+)
  RewriteRule . - [E=RU:%1]
  RequestHeader set REMOTE_USER %{RU}e
</Location>
```

Then you need to enable the following modules in your `httpd.conf` file:

Modules to enable

```
LoadModule sspi_auth_module modules/mod_auth_sspi.so
LoadModule headers_module modules/mod_headers.so
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_connect_module modules/mod_proxy_connect.so
LoadModule proxy_http_module modules/mod_proxy_http.so
LoadModule rewrite_module modules/mod_rewrite.so
```

Using API Key with HTTP-SSO Users

While HTTP-SSO provides access to Artifactory UI, it is also possible for HTTP-SSO users to generate an [API key](#) that can be used instead of a password for basic authentication or in a dedicated [REST API header](#), this is very useful when working with different clients, e.g. docker, npm, maven, etc. or using Artifactory REST API.

In order to allow HTTP-SSO users access to an API key you will need to make sure that the **"Auto Create Artifactory Users"** and **"Allow Created Users Access To Profile Page"** check boxes are checked. This means that SSO users are also saved in Artifactory database and can access their [profile page](#) in order to generate, retrieve and revoke their API key.

Smart Searches

Overview

Smart search is a feature that allows you to assemble a custom set of artifacts returned by a series of

separate searches actions. This is done by saving search results in a Stash.

The Stash provides easy access to artifacts found without having to run the series of searches again, and also provides a convenient way to perform bulk operations on the result set using the [Stash Browser](#).

Using the Stash you can save a search result, then use additional searches to add, remove and intersect new results with the original result. Effectively, you are assembling a 'shopping cart' of artifacts, which you can then manipulate as one unit.

For example, you can search for all artifacts deployed by a certain build (by build number), remove all the sources from the search results (by running another search) and promote the final result set to a public repository. Or, you can search all POMs containing a specific license and move them to a repository of approved artifacts, or attach an "approved" property to them.

Page Contents

- [Overview](#)
- [Saving Search Results in the Stash](#)
 - [View](#)
 - [Clear](#)
 - [Actions](#)
- [Stash Browser](#)
- [From Staging to Promotion](#)

Saving Search Results in the Stash

To save search results after running a search, click **Stash Results**. To save only a subset of the search results, first select the items you want to save and then click **Stash Results**. If you don't select any items, the whole result set will be saved.

The screenshot shows the JFrog Artifactory search results page. The search criteria are: Group ID 'm GAVC', Classifier 'bintray*', and Version '3*'. The search results table contains three items. The 'Stash Results' button is highlighted with a red box.

Artifact	Group ID	Artifact ID	Version	Classifier	Repository	Path	Modified
bintray-info-3.6-20150720.105830...	org.jfrog.test	bintray-info	3.6-20150720.105830-1		libs-snapshot-local	org/jfrog/test/bintray-info/3.6-SNAPSHOT/bintray-info-3.6-2...	20-07-15 10:59:05 UTC
bintray-info-3.6-20150720.105830...	org.jfrog.test	bintray-info	3.6-20150720.105830-1		libs-snapshot-local	org/jfrog/test/bintray-info/3.6-SNAPSHOT/bintray-info-3.6-2...	20-07-15 10:59:05 UTC
bintray-multi-3.6-20150720.105830...	org.jfrog.test	bintray-multi	3.6-20150720.105830-1		libs-snapshot-local	org/jfrog/test/bintray-multi/3.6-SNAPSHOT/bintray-multi-3...	20-07-15 10:58:47 UTC


Once you have items stored in the stash, Artifactory displays the number of items stored and offers several actions you can perform.

The screenshot shows the JFrog Artifactory search results page with the same search criteria as above. The 'View', 'Clear', and '3 Items' buttons are highlighted with a red box.

Artifact	Group ID	Artifact ID	Version	Classifier	Repository	Path	Modified
bintray-info-3.6-20150720.105830...	org.jfrog.test	bintray-info	3.6-20150720.105830-1		libs-snapshot-local	org/jfrog/test/bintray-info/3.6-SNAPSHOT/bintray-info-3.6-2...	20-07-15 10:59:05 UTC
bintray-info-3.6-20150720.105830...	org.jfrog.test	bintray-info	3.6-20150720.105830-1		libs-snapshot-local	org/jfrog/test/bintray-info/3.6-SNAPSHOT/bintray-info-3.6-2...	20-07-15 10:59:05 UTC
bintray-multi-3.6-20150720.105830...	org.jfrog.test	bintray-multi	3.6-20150720.105830-1		libs-snapshot-local	org/jfrog/test/bintray-multi/3.6-SNAPSHOT/bintray-multi-3...	20-07-15 10:58:47 UTC

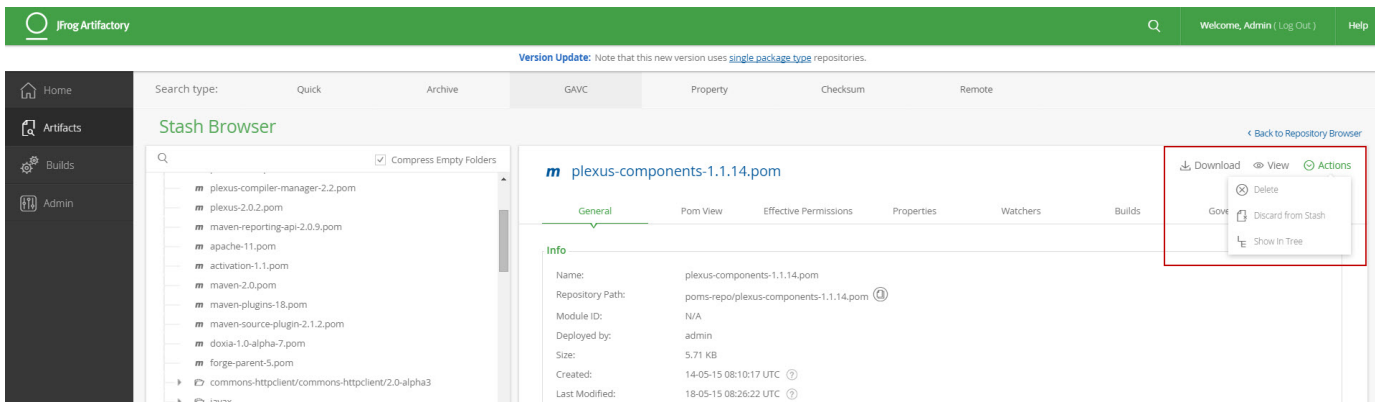
View

This displays the Stash Browser showing all items currently stored in the stash

Clear	Remove all items from the stash
Actions 	<p>Add: Adds to the stash items found in the current result set that are not already stored in the stash</p> <p>Subtract: Items found in the current search result set, that are also in the stash, are subtracted (i.e. removed) from the stash</p> <p>Intersect: Items that are in the intersection of the current search results and the current stash contents are kept in the stash. All other items are removed.</p>

Stash Browser

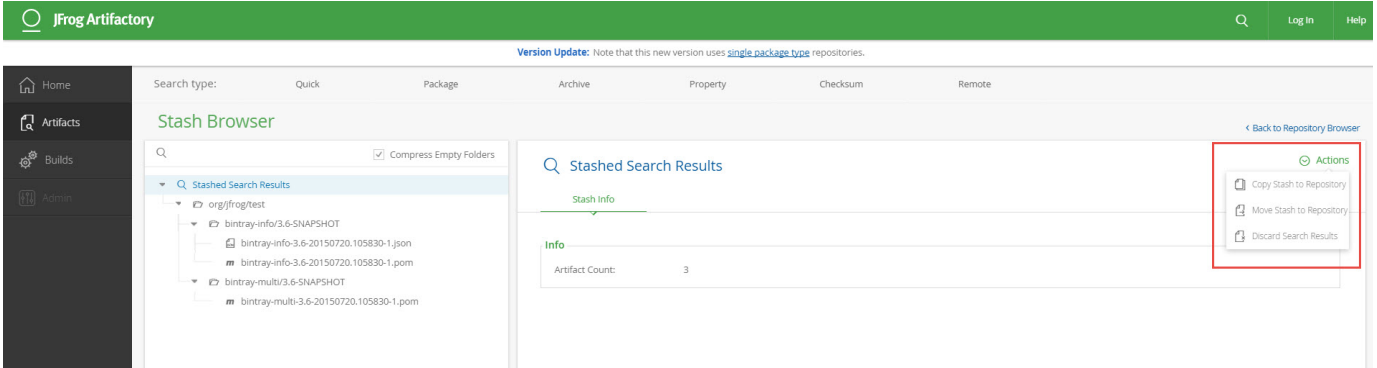
The stash browser displays all items that are in the stash. You can browse through the items and view relevant information corresponding to the item type just like you would in the [Tree Browser](#).



If you select one of the items in the stashed search results tree, the specific information panel relevant to the selected item is displayed. The **Actions** available are:

Delete	Delete the item.
Discard from Stash	Remove the item from the stash without deleting it.
Show in Tree	Display the item in the Artifact Tree Browser.
View	View the contents of the file
Download	Download the artifact or folder

If you are on the root **Stashed Search Results** item, you can perform bulk actions on all the contents of the stash at once.



Copy Stash to Repository	Copies the entire stash contents to a repository
Move Stash to Repository	Move the entire stash contents from their current location to a repository
Discard search results	Removes all items from the stash without deleting them.

On the root **Stashed Search Results** item you can also perform an export of the entire stash in the same way you would [export a repository](#).

To go back to the Artifacts Tree Browser, click **Back to Repository Browser**.

From Staging to Promotion

For more detailed information about using Smart Searches for powerful, yet simple, promotion support please see [this blog entry](#).

SSH Integration

Overview

From version 4.4, Artifactory supports SSH authentication for Git LFS and the **JFrog CLI** using RSA public and private keys. This allows these tools to exchange sensitive information with the Artifactory server that is authenticated via SSH.

There are two main facets of SSH authentication:

Server authenticates itself to the client

The server must be authenticated before you send it any confidential data. For example, you should not authenticate a user to the server with the user's password before the server has been authenticated. The server is authenticated in the following manner.

When the SSH connection is established, the server sends its public key to the client, and the client matches the key to a list of known public keys stored in a *known_hosts* file. (Before the first ever connection to the server, you must obtain the server's public key by some other means and add it to the *known_hosts* file manually). This verifies that the server is indeed the owner of the stored public key, since only that server will have the corresponding private key. It also verifies that the server is known (and not an imposter) since its public key is stored in the *known_hosts* file.

User authenticates itself to the server

This process mirrors the process of the server being authenticated to the client. The user must first provide his public key to the server which stores it in the user's account authorization list. Then, when the user tries to log in, the server sends the user back his public key, and the user must show that he holds the corresponding private key.

Limitation
SSH is not supported if using [Artifactory Saas](#) cloud service.

Page Contents

- [Overview](#)
- [Configuring SSH](#)
 - [Configuring Server Authentication](#)
 - [Configuring User Authentication](#)

- [Configuring the Client](#)

Configuring SSH

To configure SSH authentication, you need to execute the following main steps:

1. [Configure Server Authentication](#)
2. [Configure User Authentication](#)
3. [Configure the Git LFS or CLI Client](#)

Configuring Server Authentication

In this step you will configure Artifactory's SSH authentication parameters. First you need to generate an SSH key pair for Artifactory. For example, on a Linux-based system, you could execute the following command:

```
ssh-keygen -t rsa -C "server@domain.com"
```

Then, to configure Artifactory for SSH authentication, in the **Admin** module, select **Security | SSH Server** and fill in the required fields.

SSH Server Configuration

SSH Server Settings

Enable SSH Authentication

Port *

1339

Custom URL Base * ?

http://10.10.10.10/artifactory

Server Keys

Public key : No public key installed

Drop file here or [Select file](#)

Upload

Private key : No private key installed

Drop file here or [Select file](#)

Upload

Enable SSH Authentication	When checked, SSH authentication is enabled
Port	The port that should be used for an SSH connection

Custom URL Base	The Custom URL Base that should be used for SSH connections. Note that this is the same Custom URL Base configured in the Admin module under Configuration General .
Public key/Private key	The key pair used for authentication

Configuring User Authentication

In this step, you will configure Artifactory with your public key so that you may be authenticated when sending requests to Artifactory from the Git LFS client or from the Artifactory CLI.

First, you need to generate a key pair. For example, on a Linux-based system, you could execute the following command:

```
ssh-keygen -t rsa -C "USER@domain.com"
```

Your public and private keys should be created under the `~/ .ssh` folder.

Don't forget to update your public key

Update your public key under the [SSH](#) section of your User Profile.

Configuring the Client

To configure your Git LFS client, please refer to [Authenticating with SSH](#).

To configure the JFrog CLI, please refer to [Authenticating with RSA Keys](#).

User Plugins

About Plugins

Artifactory Pro allows you to easily extend Artifactory's behavior with your own plugins written in [Groovy](#).

User plugins are used for running user's code in Artifactory. Plugins allow you to perform the following tasks:

- Add scheduled tasks
- Extend Artifactory with your own security realms
- Change resolution rules
- Manipulate downloaded content
- Respond to any storage events on items and properties
- Deploy and query artifacts and metadata
- Perform searches
- Query security information
- Invoke custom commands via REST
- Execute custom promotion logic
- Provide information and strategies for [Artifactory's Build Servers Plugins](#).

During the development phase, you can change plugin source files and have your plugins redeployed

on-the-fly. You can even debug the plugin code using your favorite IDE.

Groovy Version

Groovy 2.4 is supported

Page Contents

- About Plugins
- Deploying Plugins
 - Reloading Plugins
 - Auto Reload
 - Reloading Plugins via REST API
 - Plugins Lib Directory
- Removing Plugins
- Retrieving Plugin Source Code
- Writing Plugins
 - The Artifactory Public API (PAPI)
 - Globally Bound Variables
 - Plugin Execution Points
 - Execution Context
 - Including AQL Queries
- Plugin Template Source
 - General Info
 - Download
 - Storage
 - Jobs
 - Executions
 - Realms
 - Build
 - Promotions
 - Staging
 - Replication
- Controlling Plugin Log Level
- Sample Plugin

Deploying Plugins

Place your plugin files under `${ARTIFACTORY_HOME}/etc/plugins`.

Artifactory HA Plugins Directory

If you are working with a [High Availability](#) cluster your user plugins should be added to the **primary** node under:

```
${ARTIFACTORY_HOME}/etc/plugins
```

And they will be propagated to the entire cluster.

Any file name ending with `.groovy` is loaded on startup. You can have multiple plugin files which are loaded in alphabetical order. Callbacks defined in plugins are called by the order they were loaded.

Reloading Plugins

By default, plugins are not reloaded after Artifactory has started-up. You can configure Artifactory to automatically detect plugin changes on disk or new plugin files and automatically reload them in runtime (plugin removals are not detected), or reload plugins using the REST API.

Auto Reload

To automatically reload plugins that have changed, set the number of seconds to check for plugin updates to a number greater than 0, by changing the following property in `${ARTIFACTORY_HOME}/etc/artifactory.system.properties`, or by specifying the property with `-D` to the JVM running Artifactory:

```
artifactory.plugin.scripts.refreshIntervalSecs=0
```

NOTE! that deleting or renaming plugin *files* while auto-reloading is active is not fully supported and requires an Artifactory restart.

Disabling Plugin Reloading for Production

Ensure plugin auto-reloading is disabled in a production environment.

Reloading Plugins via REST API

You can reload plugins using the [Reload Plugins REST API](#) endpoint.

Plugins Lib Directory

If your plugin requires any external dependencies, you can place them under the `${ARTIFACTORY_HOME}/etc/plugins/lib` directory.

Removing Plugins

To remove a plugin, simply delete it from the `${ARTIFACTORY_HOME}/etc/plugins` directory.

Removing Plugins from an Artifactory HA Cluster

To remove a plugin from a [High Availability](#) cluster you only need to delete the plugin file from the **master** node.

The deletion event is then propagated to all other nodes in the cluster and Artifactory will delete the respective file from each cluster node automatically.

Retrieving Plugin Source Code

You can retrieve the Groovy source code of a user plugin using the [Retrieve Plugin Code REST API](#) endpoint

Writing Plugins


Artifactory plugins are written as Groovy scripts in regular files and have a simple DSL to wrap users code in closures inside well-known extension points.

Scripts have a couple of helper objects that are globally bound (see the plugin script template).

Naming conventions

Note that Groovy scripts must follow the same [naming conventions](#) as those specified for Java.

The Artifactory Public API (PAPI)

Scripts have access to the full classpath of Artifactory, however, the only API supported for plugins is the  **Artifactory Public API**, defined in the `artifactory-papi.jar`.

The `artifactory-papi.jar` can be found under `WEB-INF/lib` folder inside the `artifactory.war`.

Please see the [Plugin Code Template](#) and [Sample Plugin](#) below for more details.

IDE code completion

All major IDEs have good Groovy editing and debugging capabilities.

In order to make your developing experience complete, we provide support for our own DSL for IntelliJ IDEA. IntelliJ IDEA's [Groovy DSL script](#) for Artifactory User Plugins can be found in our [GitHub repo](#). [Eclipse DSLD file](#) is also available courtesy of James Carnegie.

Globally Bound Variables

Variable Name	Variable Type	Comments
---------------	---------------	----------

log	<code>org.slf4j.Logger</code>	Writes to Artifactory log logger name is the name of the script file
repositories	<code>org.artifactory.repo.Repositories</code>	Allows queries and operations on repositories and artifacts
security	<code>org.artifactory.security.Security</code>	Provides information about current security context, (e.g. current user and her permissions)
searches	<code>org.artifactory.search.Searches</code>	API for searching for artifacts and builds Since 2.3.4
builds	<code>org.artifactory.build.Builds</code>	Allows CRUD operations on builds Since 2.6

Closure Variables

Note! Declaring your own closure variables using the Groovy 'def' keyword is considered best practice. Avoiding the "def" keyword is risky in terms of variable scoping, and will result in the variable being scoped globally, making it accessible from other closure executions.

Plugins Repository

Enhancing Artifactory with user plugins is community-driven effort.

If you are looking to go beyond Artifactory's out-of-the-box functionality take a look at [already contributed plugins on GitHub](#), you might find what you are thinking about. If not, your contribution is very welcome!

Plugin Execution Points

The following table summarizes the available execution points. For more details about specific plugin look follow the section links.

Plugin Type	Code block name	When executed	Description
Download			
Event Callback (with return values)	<code>altResponse</code>	On any download	Provide an alternative response, by setting a success/error status code value and an optional error message or by setting new values for the inputStream and size context variables (For succeeded resolutions).
	<code>altRemotePath</code>	When reaching out to remote repositories	Provides an alternative download path under the same remote repository, by setting a new value to the path variable.
	<code>altRemoteContent</code>	After fetching content from remote repositories	Provide an alternative download content, by setting new values for the inputStream and size context variables.
	<code>afterDownloadError</code>	After failing during content fetching from remote repositories	Provide an alternative response, by setting a success/error status code value and an optional error message or by setting new values for the inputStream and size context variables (For failed resolutions).
Event Callback (without return value)	<code>beforeRemoteDownload</code>	Before fetching content from remote repositories	Handle before remote download events.
	<code>afterRemoteDownload</code>	After fetching content from remote repositories	Handle after remote download events.
	<code>beforeDownload</code>	On any download	Handle before download events.
	<code>afterDownload</code>	On any download	Handle after download events
	<code>beforeDownloadRequest</code>	On any download	Handle before download request events, executed before Artifactory starts to handle the original client request, useful for intercepting expirable resources (other than the default ones like maven-metadata.xml).
Storage			

Event Callback (without return value)	before/after Create, Delete, Move, Copy, PropertyCreate, PropertyDelete	Before / After selected storage operation	Handle events before and after Create, Delete, Move and Copy operations
Jobs			
Scheduled execution	any valid Groovy (Java) literal as execution name	According to provided interval/delay or cron expression	Job runs are controlled by the provided interval or cron expression, which are mutually exclusive. The actual code to run as part of the job should be part of the job's closure.
Executions			
User-driven execution	any valid Groovy (Java) literal as execution name	By REST call	External executions are invoked via REST requests.
Realms			
Event Callback (without return value)	any valid Groovy (Java) literal as realm name with nested blocks: authenticate userExists	During user authentication	Newly added realms are added before any built-in realms (Artifactory internal realm, LDAP, Crowd etc.). User authentication will be attempted against these realms first, by the order they are defined.
Build			
Event Callback (without return value)	beforeSave	Before the build info is saved in Artifactory	Handle before build info save events
	afterSave	After the build info is saved in Artifactory	Handle after build info save events
Promotions			
User or build server driven execution	any valid Groovy (Java) literal as promotion name	By REST call	Promotes integration (a.k.a. snapshot) build to be a release invoking any code associated with it.
Staging Strategy			
build server driven execution	any valid Groovy (Java) literal as staging strategy name	During build server driven staging build configuration	The strategy provides the build server with the following information: <ul style="list-style-type: none"> • How the artifacts in the staged build should be versioned; • How the artifacts in the next integration build should be versioned; • Should the build server create a release branch/tag/stream in VCS and how it should be called; • To which repository in Artifactory the built artifacts should be submitted.
Replication			
Event callback (with return value)	beforeFileReplication	Before file is replicated	Handle before file replication events. File replication can be skipped.
	beforeDirectoryReplication	Before directory is replicated	Handle before directory replication events. Directory replication can be skipped.
	beforeDeleteReplication	Before file/directory is deleted	Handle before file or directory are deleted.
	beforePropertyReplication	Before properties are replicated	Handle properties replication.

Execution Context

The **Download**, **Storage**, **Execution** and **Build** plugin types are executed under the identity of the user request that triggered them.

It is possible to force a block of plugin code to execute under the "system" role, which is not bound to any authorization rules and can therefore perform actions that are otherwise forbidden for the original user.

To run under the "system" role wrap your code with the `asSystem` closure:

```

... someCode ...

asSystem {
  //This code runs as the system role
}

... someOtherCode ...

```

The **Realm** and **Job** plugin types already execute under the "system" role. This cannot be changed.

Including AQL Queries

User plugins may include [AQL queries](#) opening up the full set of search capabilities that AQL has to offer. AQL queries are implemented within the Searches object as shown in the example below.

```

import org.artifactory.repo.RepoPathFactory
import org.artifactory.search.Searches
import org.artifactory.search.aql.AqlResult

executions {
  gemPropsPopulator() {
    def repoKey = "gem-local"
    ((Searches) searches).aql(
      "items.find({" +
        "\"repo\": \"" + repoKey + "\", " +
        "\"\${or}\":[" +
        "{\"property.key\":{\"\${ne}\":\"gem.name\"}}, " +
        "{\"property.key\":{\"\${ne}\":\"gem.version\"}} " +
        "])" +
        ".include(\"path\", \"name\")" ) {
      AqlResult result ->
        result.each {
          ...
          ...
          ...
        }
      }
    }
  }
}

```

Plugin Template Source

General Info

▼ [General info...](#)

```

/*
 * Copyright (C) 2011 JFrog Ltd.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
 * implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

/**
 *
 * Globally bound variables:
 *
 * log (org.slf4j.Logger)
 * repositories (org.artifactory.repo.Repositories)
 * security (org.artifactory.security.Security)
 * searches (org.artifactory.search.Searches) [since: 2.3.4]
 * builds (org.artifactory.build.Builds) [since 2.5.2]
 *
 * ctx (org.artifactory.spring.InternalArtifactoryContext) - NOT A
 * PUBLIC API - FOR INTERNAL USE ONLY!
 */

```

Download

▼ [Handling and manipulating "download" events...](#)

```

download {

    /**
     * Provide an alternative response, by one of the following methods:
     * (1) Setting a success/error status code value and an optional error
     * message.
     * (2) Provide an alternative download content, by setting new values
     * for the inputStream and size context variables.
     *
     * Note that, unless specifically handled, checksum requests for
     * altered responses will return the checksum of the
     * original resource, which may not match the checksum of the
     * alternate response.
     *
     * Will not be called if the response is already committed (e.g. a
     * previous error occurred).
     * Currently called only for GET requests where the resource was

```

```

found.
*
* Context variables:
* status (int) - a response status code. Defaults to -1 (unset).
* message (java.lang.String) - a text message to return in the
response body, replacing the response content.
*
*                               Defaults to null.
* inputStream (java.io.InputStream) - a new stream that provides the
response content. Defaults to null.
* size (long) - the size of the new content (helpful for clients
processing the response). Defaults to -1.
* headers (java.util.Map<String,String>) - Map containing the extra
headers to override or add if not exists to the response.
*
* Usage example:
* headers = ["ExtraHeader":"SpecialHeader"]
*
*
* Closure parameters:
* request (org.artifactory.request.Request) - a read-only parameter
of the request.
* responseRepoPath (org.artifactory.repo.RepoPath) - a read-only
parameter of the response RepoPath (containing the
*                               physical
repository the resource was found in).
*/
altResponse { request, responseRepoPath ->
}

/**
* Provides an alternative download path under the same remote
repository, by setting a new value to the path
* variable.
*
* Context variables:
* path (java.lang.String) - the new path value. Defaults to the
originalRepoPath's path.
*
* Closure parameters:
* repoPath (org.artifactory.repo.RepoPath) - a read-only parameter of
the original request RepoPath.
*/
altRemotePath { repoPath ->
}

/**
* Provide an alternative download content, by setting new values for
the inputStream and size context variables.
*
* Context variables:
* inputStream (java.io.InputStream) - a new stream that provides the
response content. Defaults to null.

```

```

    * size (long) - the size of the new content (helpful for clients
processing the response). Defaults to -1.
    *
    * Closure parameters:
    * repoPath (org.artifactory.repo.RepoPath) - a read-only parameter of
the original request RepoPath.
    */
    altRemoteContent { repoPath ->
    }

/**
    * In case of resolution error provide an alternative response, by
setting a success/error status code value and an optional error message.
    * Will not be called if the response is already committed (e.g. a
previous error occurred).
    * As opposite to altResponse, called only for GET requests during
which error occurred (e.g. 404 - not found, or 409 - conflict).
    *
    * Context variables:
    * status (int) - a response error status code (may be overridden in
the plugin).
    * message (java.lang.String) - a response error message (may be
overridden in the plugin).
    * inputStream (java.io.InputStream) - a new stream that provides the
response content. Defaults to null.
    * size (long) - the size of the new content (helpful for clients
processing the response). Defaults to -1.
    *
    * Closure parameters:
    * request (org.artifactory.request.Request) - a read-only parameter
of the request.
    */
    afterDownloadError { request ->
    }

/**
    * Handle before remote download events.
    *
    * Context variables:
    * headers (java.util.Map<String,String>) - Map containing the extra
headers to insert into the remote server request
    *
    * Usage example:
    * headers = ["ExtraHeader":"SpecialHeader"]
    *
    * Note: The following cannot be used as extra headers and Artifactory
will always override them:
    * "X-Artifactory-Originated". "Origin-Artifactory", "Accept-Encoding"
    *
    * Closure parameters:
    * request (org.artifactory.request.Request) - a read-only parameter
of the request. [since: 2.3.4]
    * repoPath (org.artifactory.repo.RepoPath) - a read-only parameter of

```

```

the original request RepoPath.
    */
    beforeRemoteDownload { request, repoPath ->
    }

/**
 * Handle after remote download events.
 *
 * Closure parameters:
 * request (org.artifactory.request.Request) - a read-only parameter
of the request. [since: 2.3.4]
 * repoPath (org.artifactory.repo.RepoPath) - a read-only parameter of
the original request RepoPath.
 */
    afterRemoteDownload { request, repoPath ->
    }

/**
 * Handle before local download events.
 *
 * Closure parameters:
 * request (org.artifactory.request.Request) - a read-only parameter
of the request.
 * responseRepoPath (org.artifactory.repo.RepoPath) - a read-only
parameter of the response RepoPath (containing the
 *                                     physical
repository the resource was found in).
 */
    beforeDownload { request, responseRepoPath ->
    }

/**
 * Handle before any download events, at this point the request passed
all of Artifactory's filters (authentication etc) and is about to reach
the repositories.
 *
 * Context variables:
 * expired (boolean) - Mark the requested resource as expired.
Defaults to false (unset).
 *                                     An expired resource is one that it's (now() -
(last updated time)) time is higher than the repository retrieval cache
period milliseconds.
 *                                     Setting this option to true should be treated
with caution, as it means both another database hit (for updating the
last updated time)
 *                                     as well as network overhead since if the resource is expired,
a remote download will occur to re-download it to the cache.
 *                                     A common implementation of this extension point is to check if
the resource comply with a certain pattern (for example: a *.json file)
 *                                     AND the original request was to the remote repository (and not
directly to it's cache)
 *                                     AND a certain amount of time has passed since the last expiry
check (to minimize DB hits).

```



```
*           See our public GitHub for an example here:
https://github.com/JFrogDev/artifactory-user-plugins/blob/master/download/beforeDownloadRequest/beforeDownloadRequest.groovy
*
* modifiedRepoPath (org.artifactory.repo.RepoPath)
*           Forces Artifactory to store the file at the specified
repository path in the remote cache.
*           See our public GitHub for an example here:
https://github.com/JFrogDev/artifactory-user-plugins/blob/master/download/modifyMD5File/ModifyMD5FileTest.groovy
* Closure parameters:
* request (org.artifactory.request.Request) - a read-only parameter
of the request.
* repoPath (org.artifactory.repo.RepoPath) - a read-only parameter
of the response RepoPath (containing the
*                                           physical
repository the resource was found in).
*/
```

```
beforeDownloadRequest { request, repoPath ->
}
}
```

Storage

✓ Handling and manipulating "storage" events...

If you want to abort an action, you can do that in 'before' methods by throwing a runtime `org.artifactory.exception.CancelException` with an error message and a proper http error code.

```
storage {

  /**
   * Handle before create events.
   *
   * Closure parameters:
   * item (org.artifactory.fs.ItemInfo) - the original item being created.
   */
  beforeCreate { item ->
  }

  /**
   * Handle after create events.
   *
   * Closure parameters:
   * item (org.artifactory.fs.ItemInfo) - the original item being created.
   */
  afterCreate { item ->
  }

  /**
   * Handle before delete events.
   *
   * Closure parameters:
   * item (org.artifactory.fs.ItemInfo) - the original item being being
   deleted.
   */
  beforeDelete { item ->
  }

  /**
   * Handle after delete events.
   *
   * Closure parameters:
   * item (org.artifactory.fs.ItemInfo) - the original item deleted.
   */
  afterDelete { item ->
  }

  /**
```

```
* Handle before move events.
*
* Closure parameters:
*
* item (org.artifactory.fs.ItemInfo) - the source item being moved.
* targetRepoPath (org.artifactory.repo.RepoPath) - the target repoPath
for the move.
*/
beforeMove { item, targetRepoPath, properties ->
}

/**
* Handle after move events.
*
* Closure parameters:
* item (org.artifactory.fs.ItemInfo) - the source item moved.
* targetRepoPath (org.artifactory.repo.RepoPath) - the target repoPath
for the move.
*/
afterMove { item, targetRepoPath, properties ->
}

/**
* Handle before copy events.
*
* Closure parameters:
* item (org.artifactory.fs.ItemInfo) - the source item being copied.
* targetRepoPath (org.artifactory.repo.RepoPath) - the target repoPath
for the copy.
*/
beforeCopy { item, targetRepoPath, properties ->
}

/**
* Handle after copy events.
*
* Closure parameters:
* item (org.artifactory.fs.ItemInfo) - the source item copied.
* targetRepoPath (org.artifactory.repo.RepoPath) - the target repoPath
for the copy.
*/
afterCopy { item, targetRepoPath, properties ->
}

/**
* Handle before property create events.
*
* Closure parameters:
* item (org.artifactory.fs.ItemInfo) - the item on which the property
is being set.
* name (java.lang.String) - the name of the property being set.
* values (java.lang.String[]) - A string array of values being assigned
to the property.
```

```
*/
beforePropertyCreate { item, name, values ->
}
/**
* Handle after property create events.
*
* Closure parameters:
* item (org.artifactory.fs.ItemInfo) - the item on which the property
has been set.
* name (java.lang.String) - the name of the property that has been set.

* values (java.lang.String[]) - A string array of values assigned to
the property.
*/
afterPropertyCreate { item, name, values ->
}
/**
* Handle before property delete events.
*
* Closure parameters:
* item (org.artifactory.fs.ItemInfo) - the item from which the property
is being deleted.
* name (java.lang.String) - the name of the property being deleted.
*/
beforePropertyDelete { item, name ->
}
/**
* Handle after property delete events.
*
* Closure parameters:
* item (org.artifactory.fs.ItemInfo) - the item from which the property
has been deleted.
* name (java.lang.String) - the name of the property that has been
deleted.
*/
```

```
afterPropertyDelete { item, name ->
}
}
```

Jobs

▼ Defining scheduled jobs...

```
jobs {

  /**
   * A job definition.
   * The first value is a unique name for the job.
   * Job runs are controlled by the provided interval or cron
   expression, which are mutually exclusive.
   * The actual code to run as part of the job should be part of the
   job's closure.
   *
   * Parameters:
   * delay (long) - An initial delay in milliseconds before the job
   starts running (not applicable for a cron job).
   * interval (long) - An interval in milliseconds between job runs.
   * cron (java.lang.String) - A valid cron expression used to schedule
   job runs (see:
   http://www.quartz-scheduler.org/docs/tutorial/TutorialLesson06.html)
   */

  myJob(interval: 1000, delay: 100) {
  }

  mySecondJob(cron: "0/1 * * * * ?") {
  }
}
```

Executions

▼ Defining external executions...

```
curl -X GET -v -u admin:password
"http://localhost:8080/artifactory/api/plugins/execute/myExecution?param
s=msg=And+the+result+is:|no1=10|no2=15&async=0"
```

```

executions {

    /**
     * An execution definition.
     * The first value is a unique name for the execution.
     *
     * Context variables:
     * status (int) - a response status code. Defaults to -1 (unset). Not
     applicable for an async execution.
     * message (java.lang.String) - a text message to return in the
     response body, replacing the response content.
     *
     Defaults to null. Not applicable for
     an async execution.
     *
     * Plugin info annotation parameters:
     * version (java.lang.String) - Closure version. Optional.
     * description (java.lang.String) - Closure description. Optional.
     * httpMethod (java.lang.String, values are GET|PUT|DELETE|POST) -
     HTTP method this closure is going
     * to be invoked with. Optional (defaults to POST).
     * params (java.util.Map<java.lang.String, java.lang.String>) -
     Closure default parameters. Optional.
     * users (java.util.Set<java.lang.String>) - Users permitted to query
     this plugin for information or invoke it.
     * groups (java.util.Set<java.lang.String>) - Groups permitted to
     query this plugin for information or invoke it.
     *
     * Closure parameters:
     * params (java.util.Map) - An execution takes a read-only key-value
     map that corresponds to the REST request
     * parameter 'params'. Each entry in the map contains an array of
     values. This is the default closure parameter,
     * and so if not named it will be "it" in groovy.
     * ResourceStreamHandle body - Enables you to access the full input
     stream of the request body.
     * This will be considered only if the type ResourceStreamHandle is
     declared in the closure.
     */

    myExecution(version:version, description:description, httpMethod:
'GET', users:[], groups:[], params[:]) { params ->
    }

    execWithBody(version:version, description:description, httpMethod:
'GET', users:[], groups:[], params[:]) { params, ResourceStreamHandle
body ->
    }

}

```

Realms

▼ [Management of security realms...](#)

Realms defined here are added before any built-in realms (Artifactory internal realm, LDAP, Crowd etc.). User authentication will be attempted against these realms first, by the order they are defined.

```

realms {

  /**
   * A security realm definition.
   * The first value is a unique name for the realm.
   *
   * Closure parameters:
   * autoCreateUsers (boolean) - Whether to automatically create users
   in Artifactory upon successful login. Defaults to
   * true. When false, the user will be transient and his privileges
   will be managed according to permissions defined for auto-join groups.
   * realmPolicy (org.artifactory.security.RealmPolicy): (Optional) - If
   included with value RealmPolicy.ADDITIVE, plugin will be executed only
   if the user has previously been authenticated, and allows enrichment of
   the authenticated
   * user with additional data.
   * See our public GitHub for an example here:
   https://github.com/JFrogDev/artifactory-user-plugins/blob/master/security/synchronizeLdapGroups/synchronizeLdapGroups.groovy
   */

  myRealm(autoCreateUsers: true, realmPolicy: RealmPolicy.ADDITIVE) {
    /**
     * Implementation should return true/false as the result of the
     authentication.
     *
     * Context variables:
     * groups (java.lang.String[]) - An array of groups that the
     authenticated user should be associated with (since 3.0.2).
     * user (org.artifactory.security.User) - The authenticated user.
     *
     * Closure parameters:
     * username (java.lang.String) - The username
     * credentials (java.lang.String) - The password
     */
    authenticate { username, credentials ->
    }

    /**
     * Implementation should return true if the user is found in the
     realm.
     * Closure parameters:
     * username (java.lang.String) - The username
     */
    userExists { username ->
    }
  }
}

```


▼ Handling "Build Info" events...

```
build {  
  
    /**  
     * Handle before build info save events  
     *  
     * Closure parameters:  
     * buildRun (org.artifactory.build.DetailedBuildRun) - Build Info  
model to be saved. Partially mutable.  
     */  
    beforeSave { buildRun ->  
    }  
  
    /**  
     * Handle after build info save events  
     *  
     * Closure parameters:  
     * buildRun (org.artifactory.build.DetailedBuildRun) - Build Info that  
was saved. Partially mutable.  
     */  
    afterSave { buildRun ->  
    }  
}
```

Promotions

▼ Defining REST executable build promotion operations...

```

promotions {

    /**
     * A REST executable build promotion definition.
     *
     * Context variables:
     * status (int) - a response status code. Defaults to -1 (unset).
     * message (java.lang.String) - a text message to return in the
response body, replacing the response content. Defaults to null.
     *
     * Plugin info annotation parameters:
     * version (java.lang.String) - Closure version. Optional.
     * description (java.lang.String) - Closure description. Optional.
     * params (java.util.Map<java.lang.String, java.lang.String>) -
Closure parameters. Optional.
     * users (java.util.Set<java.lang.String>) - Users permitted to query
this plugin for information or invoke it.
     * groups (java.util.Set<java.lang.String>) - Groups permitted to
query this plugin for information or invoke it.
     *
     * Closure parameters:
     * buildName (java.lang.String) - The build name specified in the REST
request.
     * buildNumber (java.lang.String) - The build number specified in the
REST request.
     * params (java.util.Map<java.lang.String,
java.util.List<java.lang.String>>) - The parameters specified in the
REST request.
     */
    promotionName(version, description, users, groups, params) {
        buildName, buildNumber, params ->
    }
}

```

Staging

✓ [Defining REST retrievable build staging strategy construction...](#)

```

/**
 * Set of staging strategy definitions to be used by the build server
during staging process.
 * The strategy provides the build server with the following
information:
 * 1. How the artifacts in the staged build should be versioned;
 * 2. How the artifacts in the next integration build should be
versioned;
 * 3. Should the build server create a release branch/tag/stream in VCS
and how it should be called;
 * 4. To which repository in Artifactory the built artifacts should be
submitted.
 *
 * This user plugin is called by the build server using REST call.
 */
staging {

  /**
 * A build staging strategy definition.
 *
 * Closure delegate:
 * org.artifactory.build.staging.BuildStagingStrategy - The strategy
that's to be returned.
 *
 * Plugin info annotation parameters:
 * version (java.lang.String) - Closure version. Optional.
 * description (java.lang.String - Closure description. Optional.
 * params (java.util.Map<java.lang.String, java.lang.String>) - Closure
parameters. Optional.
 * users (java.util.Set<java.lang.String>) - Users permitted to query
this plugin for information or invoke it.
 * groups (java.util.Set<java.lang.String>) - Groups permitted to query
this plugin for information or invoke it.
 *
 * Closure parameters:
 * buildName (java.lang.String) - The build name specified in the REST
request.
 * params (java.util.Map<java.lang.String,
java.util.List<java.lang.String>>) - The parameters specified in the
REST request.
 */
  strategyName(version, description, users, groups, params) { buildName,
params ->
  }
}
}

```

Replication

▼ [Handling and filtering replication events \(since version 3.0.4\)...](#)

```

replication {
  /**

```

```

    * Handle before file replication events.
    *
    * Context variables:
    * skip (boolean) - whether to skip replication for the current
item. Defaults to false. Set to true to skip replication.
    * targetInfo (org.artifactory.addon.replication.ReplicationTargetInfo)
- contains information about the replication target server
    *
    * Closure parameters:
    * localRepoPath (org.artifactory.repo.RepoPath) - the repoPath of
the item on the local Artifactory server.
    */
beforeFileReplication { localRepoPath ->
}
/**
    * Handle before directory replication events.
    *
    * Context variables:
    * skip (boolean) - whether to skip replication for the current
item. Defaults to false. Set to true to skip replication.
    * targetInfo (org.artifactory.addon.replication.ReplicationTargetInfo)
- contains information about the replication target server
    *
    * Closure parameters:
    * localRepoPath (org.artifactory.repo.RepoPath) - the repoPath of
the item on the local Artifactory server.
    */
beforeDirectoryReplication { localRepoPath ->
}
/**
    * Handle before delete replication events.
    *
    * Context variables:
    * skip (boolean) - whether to skip replication for the current
item. Defaults to false. Set to true to skip replication.
    * targetInfo (org.artifactory.addon.replication.ReplicationTargetInfo)
- contains information about the replication target server
    *
    * Closure parameters:
    * localRepoPath (org.artifactory.repo.RepoPath) - the repoPath of
the item on the local Artifactory server.
    */
beforeDeleteReplication { localRepoPath ->
}
/**
    * Handle before property replication events.
    *
    * Context variables:
    * skip (boolean) - whether to skip replication for the current
item. Defaults to false. Set to true to skip replication.
    * targetInfo (org.artifactory.addon.replication.ReplicationTargetInfo)
- contains information about the replication target server
    *

```

```
    * Closure parameters:
    * localRepoPath (org.artifactory.repo.RepoPath) - the repoPath of
the item on the local Artifactory server.
    */
beforePropertyReplication { localRepoPath ->
}
/**
 * Handle before statistics replication events.
 *
 * Context variables:
 * skip (boolean) - whether to skip replication for the current
item. Defaults to false. Set to true to skip replication.
 * targetInfo (org.artifactory.addon.replication.ReplicationTargetInfo)
- contains information about the replication target server
 *
 * Closure parameters:
 * localRepoPath (org.artifactory.repo.RepoPath) - the repoPath of
the item on the local Artifactory server.
 */
```

```
beforeStatisticsReplication { localRepoPath ->
}
}
```

Controlling Plugin Log Level

The default log level for user plugins is "warn". To change a plugin log level, add the following to `${ARTIFACTORY_HOME}/etc/logback.xml`:

```
<logger name="my-plugin">
  <level value="info"/>
</logger>
```

The logger name is the name of the plugin file without the ".groovy" extension (in the example above the plugin file name is `my-plugin.groovy`). The logging levels can be either error, warn, info, debug or trace.

Sample Plugin

Sample plugin is [available to download](#).

Watches

Overview

The Watches feature allows you to monitor selected artifacts, folders or repositories for storage events (create/delete/modify) and receive detailed email notifications on repository changes that are of interest to you.

You can add and remove Watches from the 'General' tab in the tree browser. Watches or folders intercept changes on all children. An admin can view and manage watches via the 'Watches' tab in the tree browser.

Watch notifications are aggregated at around 1 minute intervals and sent in a single email message.

All notifications respect the read permissions of the watcher on the watched item(s).

WebStart and Jar Signing

Overview

Java Web Start is a technology developed by Sun Microsystems (now Oracle) to allow you to download and run Java applications directly from your browser with one-click activation.

Java Web Start requires that any JAR downloaded is signed by the software vendor. To support this requirement, Artifactory lets you manage a set of signing keys that are used to automatically sign JAR files downloaded from a virtual repository.

For more information, please refer to the [Oracle documentation for Java Web Start](#).

Managing Signing Keys

Signing keys are managed in the **Admin** module under **Security | Signing Keys**.

Debian Signing Key

Debian signing keys are also managed on this page, however these are not related to JAR signing. For details, please refer to [Debian Signing Keys](#).

Generating JAR Signing Keys

In order to sign JAR files, you first need to create a keystore, and generate and add key pairs to it. These can be created with Oracle's `keytool` utility, that comes built into your Java Runtime Environment (JRE), by executing the following command:

```
keytool -keystore <keystore filename> -keypass
<key_password> -storepass <store_password> -alias
<store_alias> \
-genkeypair -dname "cn=<cName>, ou=<orgUnit>,
o=<orgName>, S=<stateName>, c=<country>" -validity
<days>
```

For details, please refer to the Oracle [keytool - Key and Certificate Management Tool](#) documentation.

Page Contents

- Overview
- Managing Signing Keys
 - Generating JAR Signing Keys
 - Setting Your Keystore and Keys
 - Removing a Key Pair
 - Configuring Virtual Repositories to Sign JARs

Setting Your Keystore and Keys

Before you can add a keystore, you must set the password that will be needed to make any later changes to the keystore. You will need this password to remove or update the keystore.

Set the password and click "Create". This will unlock the rest of the keystore management fields.

Change Key Store Password

Password *	Retype Password *
<input type="password" value="....."/>	<input type="password" value="....."/>
<input type="button" value="Create"/>	

Once your keystore password is set and you have created a keystore and a set of signing keys, you can add them to Artifactory.

First upload your keystore file under **Add Key-Store** and enter the keystore password. Click "Unlock"

Add Key-Store

Key-Store *	Key-Store Password *
<input type="text" value="acme-demo.store"/>	<input type="password" value="....."/>
<input type="button" value="Unlock"/>	

Once your keystore is set in Artifactory you may add key pairs under **Add Key-Pair**.

Add Key-Pair

Key-Pair Name *	Key-Pair Alias Name *
<input type="text" value="ACME"/>	<input type="text" value="acme-demo"/>
Private Key Password *	
<input type="password" value="....."/>	
	<input type="button" value="Cancel"/> <input type="button" value="Save Keypair"/>

Removing a Key Pair

To remove a key pair, simply select the key pair and click "Remove".

Remove Key-Pair

Choose a Key-Pair to Remove	<input type="button" value="Remove"/>
<input type="text" value="ACME"/>	

Configuring Virtual Repositories to Sign JARs

Once Artifactory has a keystore and key pairs, you can configure a virtual repository with the key pair you wish to use for JAR signing. This is done in the **Advanced** settings of the virtual repository configuration screen.

Edit libs-release Repository

Basic

Advanced

Artifactory Requests Can Retrieve Remote Artifacts ?

Cleanup Repository References in POMs ?

Discard active references

Key-Pair

acme

Package Management

Overview

Artifactory Pro brings the universal nature of Artifactory to full force with advanced package management for all major packaging formats in use today. As the only repository with a unique architecture that includes a filestore layer and a separate database layer, Artifactory is the only repository manager that can natively

support current package formats as well as any new format that may arise from time to time. With a paradigm of single-type-repositories, all repositories are assigned a type upon creation allowing efficient indexing to allow any client or dependency manager to work directly with Artifactory transparently as its natural repository.

Artifactory Pro currently supports the following package formats with new formats added regularly as the need arises.

Bower	Boost your front end development by hosting your own Bower components and proxying the Bower registry in Artifactory.
Chef	Enhance your capabilities for configuration management with Chef using all the benefits of a repository manager.
CocoaPods	Speed up development with Xcode and CocoaPods with fully fledged CocoaPods repositories.
Conan	Artifactory is the only secure, private repository for C/C++ packages with fine-grained access control.
Debian	Host and provision Debian packages complete with GPG signatures.
Docker	Host your own secure private Docker registries and proxy external Docker registries such as Docker Hub.
Git LFS	Optimize your workflow when working with large media files and other binary resources.
Maven	Artifactory is both a source for Maven artifacts needed for a build, and a target to deploy artifacts generated in the build process.
npm	Host your own node.js packages, and proxy remote npm repositories like npmjs.org through Artifactory.
NuGet	Host and proxy NuGet packages in Artifactory, and pull libraries from Artifactory into your various Visual Studio .NET applications.
Opkg	Optimize your work with OpenWrt using Opkg repositories. Proxy the official OpenWrt repository and cache remote <i>.ipk</i> files.
P2	Proxy and host all your Eclipse plugins via an Artifactory P2 repository, allowing users to have a single-access-point for all Eclipse updates.
PHP Composer	Provision Composer packages from Artifactory to the Composer command line tool, and access Packagist and other remote Composer metadata repositories.
Puppet	Configuration management meets repository management with Puppet repositories in Artifactory.
PyPI	Host and proxy PyPI distributions with full support for pip.
RPM	Distribute RPMs directly from your Artifactory server, acting as fully-featured YUM repository.
RubyGems	Use Artifactory to host your own gems and proxy remote gem repositories like rubygems.org .

SBT	Resolve dependencies from and deploy build output to SBT repositories when running SBT builds.
Vagrant	Securely host your Vagrant boxes in local repositories.
VCS	Consume source files packaged as binaries.

Page Contents

- [Overview](#)

Read more

- [Bower Repositories](#)
- [Chef Cookbook Repositories](#)
- [CocoaPods Repositories](#)
- [Conan Repositories](#)
- [Debian Repositories](#)
- [Docker Registry](#)
- [Git LFS Repositories](#)
- [Npm Registry](#)
- [NuGet Repositories](#)
- [Opkg Repositories](#)
- [P2 Repositories](#)
- [PHP Composer Repositories](#)
- [Puppet Repositories](#)
- [PyPI Repositories](#)
- [RPM Repositories](#)
- [RubyGems Repositories](#)
- [SBT Repositories](#)
- [Vagrant Repositories](#)
- [VCS Repositories](#)

Bower Repositories

Overview

Artifactory supports [bower](#) repositories on top its [existing support](#) for advanced artifact management.

Artifactory support for Bower provides:

1. The ability to provision Bower packages from Artifactory to the Bower command line tool from all repository types.
2. Calculation of Metadata for Bower packages hosted in Artifactory's local repositories.
3. Access to remote Bower registries (such as <http://bower.herokuapp.com>) through [Remote Repositories](#) which provide the usual proxy and caching functionality.
4. The ability to access multiple Bower registries from a single URL by aggregating them under a [Virtual Repository](#).
5. Assign access privileges according to projects or development teams.

Configuration

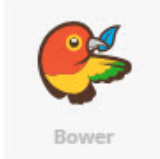
Local Repositories

To enable calculation of Bower package metadata set **Bower** to be the **Package Type** when you create your local Bower repository.

New Local Repository

Basic

Package Type *



Page Contents

- Overview
- Configuration
 - Local Repositories
 - Deploying Bower Packages
 - Remote Repositories
 - Virtual Repositories
 - Advanced Configuration
- Using the Bower Command Line
 - Using Bower Version 1.5 and above
 - Using Older Versions of Bower
- Working with Artifactory without Anonymous Access
- Cleaning Up the Local Bower Cache
- Automatically Rewriting External Dependencies
 - Rewriting Workflow
 - Using the Bower Shorth and Resolver
- Registering Bower Packages
- Viewing Individual Bower Package Information

Deploying Bower Packages

The Bower client does not provide a way to deploy packages and relies on a Git repository to host the Bower package code. To deploy a Bower package into Artifactory, you need to use Artifactory's [REST API](#) or the [Web UI](#).

A Bower package is a simple `tar.gz` file which contains your project code as well as a `bower.json` file describing the package name and version.

Usually, you will use a custom [Grunt/ Gulp](#) task to pack your project into an archive file and deploy it to Artifactory.

Version property

Make sure to include a `version` property in your `bower.json` file. You can add the property manually or by using the `bower version` command.

Remote Repositories

The public [bower registry](#) does not contain any actual binary packages; it is a simple key-value store pointing from a package name to its equivalent Git repository.

Since most of the packages are hosted in GitHub, you will want to create a [Remote Repository](#) which serves as a caching proxy for [github.com](#). If necessary, you can do the same for [bitbucket.org](#) or any other remote repository you want to access.

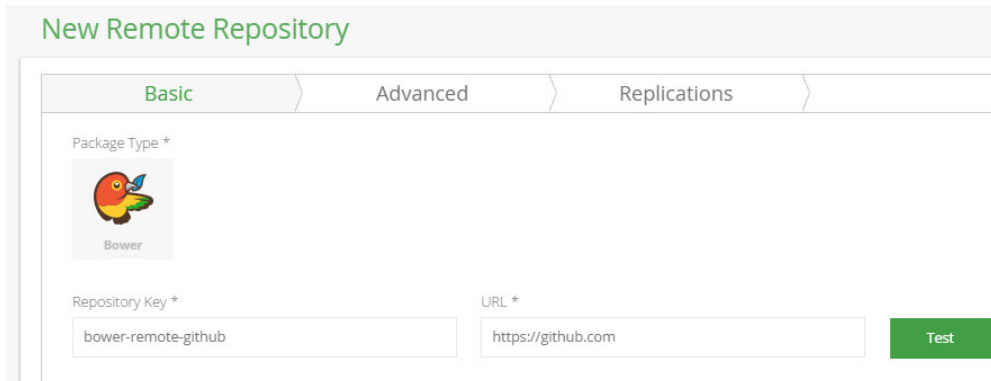
Working with Bitbucket?

If your packages are hosted on Bitbucket (formerly Stash), you need to ensure that the Bitbucket Archive Plugin is installed on your Bitbucket server.

Artifacts (such as `tar.gz` files) requested from a remote repository are cached on demand. You can remove downloaded artifacts from the remote repository cache, however you can not manually deploy artifacts to a remote repository.

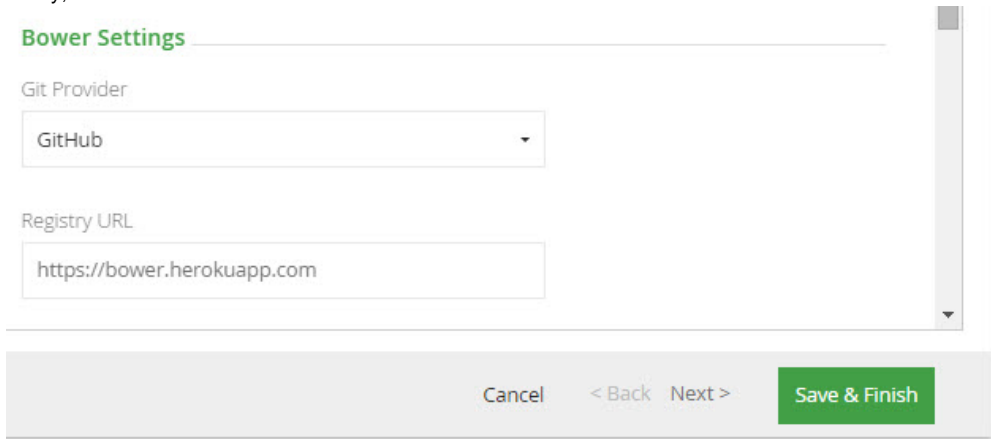
To define a remote repository to proxy `github.com` as well as the public Bower registry follow the steps below:

1. Create a new remote repository and set **Bower** to be its **Package Type**
2. Set the **Repository Key** value, and enter `https://github.com` in the **URL** field as displayed below



The screenshot shows the 'New Remote Repository' form with the 'Basic' tab selected. The 'Package Type' is set to 'Bower' (indicated by a parrot icon). The 'Repository Key' field contains 'bower-remote-github' and the 'URL' field contains 'https://github.com'. A green 'Test' button is visible to the right of the URL field.

3. In the **Bower Settings** section, select **GitHub** as the **Git Provider**. Finally, click "Save & Finish"



The screenshot shows the 'Bower Settings' section. The 'Git Provider' dropdown menu is set to 'GitHub'. The 'Registry URL' field contains 'https://bower.herokuapp.com'. At the bottom of the form, there are buttons for 'Cancel', '< Back', 'Next >', and a green 'Save & Finish' button.

Bower Registry URL

Usually, you will point the **Bower Registry URL** field at the public registry as displayed above.

However, if you are using a private bower registry or a remote Artifactory instance, simply set the same URL as configured in **URL** field.

Bower have changed their registry URL from the default configured in Artifactory. In order to resolve from the public registry, set the Registry URL to <https://registry.bower.io>.

Virtual Repositories

A Virtual Repository defined in Artifactory aggregates packages from both local and remote repositories. This allows you to access both locally hosted Bower packages and remote proxied Bower registries from a single URL defined for the virtual repository.

To create a virtual Bower repository set **Bower** to be its **Package Type**, and select the underlying local and remote bower repositories to include under the **Repositories** section.

Repositories

Available Repositories

«

<

>

»

Selected Repositories

📦 bower-local✕

📦 bower-remote-github✕

Included Repositories

bower-local

bower-remote-github

Advanced Configuration

The fields under **External Dependency Rewrite** are connected to [automatically rewriting external dependencies](#) for Bower packages that need them.

Edit bower-virtual Repository

Basic

Advanced

Artifactory Requests Can Retrieve Remote Artifacts [?](#)

External Dependency Rewrite

Enable Dependency Rewrite

Remote Repository For Cache

bower-remote

Patterns Whitelist [?](#)

New Pattern

Add

/github.com/

Enable Dependency Rewrite	When checked, automatically rewriting external dependencies is enabled.
Remote Repository For Cache	The remote repository aggregated by this virtual repository in which the external dependency will be cached.
Patterns Whitelist	<p>A white list of Ant-style path expressions that specify where external dependencies may be downloaded from. By default, this is set to <code>**</code> which means that dependencies may be downloaded from any external source.</p> <p>For example, if you wish to limit external dependencies to only be downloaded from <code>github.com</code>, you should add <code>**/github.com/**</code> (and remove the default <code>**</code> expression).</p>

Using the Bower Command Line

Bower repositories must be prefixed with `api/bower` in the path

When accessing a Bower repository through Artifactory, the repository URL must be prefixed with `api/bower` in the path. This applies to all Bower commands including `bower install` and `bower info`.

For example, if you are using Artifactory standalone or as a local service, you would access your Bower repositories using the following URL:

```
http://localhost:8081/artifactory/api/bower/<repository key>
```

Or, if you are using Artifactory SaaS, the URL would be:

```
https://<server name>.jfrog.io/<server name>/api/bower/<repository key>
```

Artifactory has been updated to work seamlessly with the latest version of the Bower client from version 1.5, and also supports older versions of Bower.

Older versions of Bower

If your version of Bower is below 1.5, please refer to [Using Older Versions of Bower](#).

Using Bower Version 1.5 and above

In order to use Bower with Artifactory you need 2 components (npm packages):

1. `bower-art-resolver` - A custom, pluggable Bower resolver which is dedicated to integrate with Artifactory.
2. `bower` - Bower version **1.5.0** and above.

Once Bower is installed, add the Artifactory Bower resolver by editing your `~/.bowerrc` configuration file

Adding a Pluggable Resolver

```
{
  "resolvers": [
    "bower-art-resolver"
  ]
}
```

Bower Documentation

For more information, please refer to the Bower documentation on [Pluggable Resolvers](#).

Replace the default registry with a URL pointing to a Bower repository in Artifactory by editing your `~/.bowerrc` configuration file (the example below uses a repository with the key `bower-repo`):

Replacing the default registry

```
{
  "registry": "http://localhost:8081/artifactory/api/bower/bower-repo"
}
```

Using the Bower Shorthand Resolver

If you want to configure the Bower Shorthand Resolver to work with Artifactory, please refer to [Bower Shorthand Resolver](#) below.

.bowerrc file location

Windows: `%userprofile%\ .bowerrc`

Linux: `~/.bowerrc`

We recommend referencing a [Virtual Repository](#) URL as a registry. This gives you the flexibility to reconfigure and aggregate other external sources and local repositories of Bower packages you deployed.

Once the Bower command line tool is configured, every `bower install` command will fetch packages from the bower repository specified above. For example:

```
$ bower install bootstrap
bower bootstrap#*      not-cached art://twbs/bootstrap#*
bower bootstrap#*      resolve art://twbs/bootstrap#*
bower bootstrap#*      extract archive.tar.gz
bower bootstrap#*      resolved art://twbs/bootstrap#e-tag:0b9cb774e1
```

Using Older Versions of Bower

Using Bower below version 1.5...

Version support

Older versions of Bower are only supported by Artifactory up to version 4.2.0.

In order to use Bower below version 1.5 with Artifactory you need 2 components (npm packages):

1. [bower-art-resolver](#) - A custom Bower resolver dedicated to integrate with Artifactory.
2. [bower-art](#) - A temporary custom Bower CLI with the pluggable resolvers mechanism currently in [pending pull request](#).

The `bower-art` package is a peer dependency of `bower-art-resolver`. Therefore, both can be easily installed with:

```
npm install -g bower-art-resolver
```

Use `bower-art` instead of `bower`

While Artifactory support for Bower is in Beta, after installing the required components, you need to execute `bower-art` instead of each `bower` command.

For example, use `bower-art install <pkg>` instead of `bower install <pkg>`

Updating Resolver

In order to update Artifactory resolver, please **uninstall** the "bower-art" npm package first, and then install the resolver. This step is necessary because npm doesn't update peer dependencies.

Once `bower-art` is installed, replace the default registry with a URL pointing to a Bower repository in Artifactory by editing your `~/.bowerrc` configuration file (the example below uses a repository with the key `bower-repo`):

Replacing the default registry

```
{
  "registry": "http://localhost:8081/artifactory/api/bower/bower-repo"
}
```

`.bowerrc` file location

Windows: `%userprofile%\ .bowerrc`

Linux: `~/.bowerrc`

We recommend referencing a [Virtual Repository](#) URL as a registry. This gives you the flexibility to reconfigure and aggregate other external sources and local repositories of Bower packages you deployed.

Once the Bower command line tool is configured, every `bower-art install` command will fetch packages from the bower repository specified above. For example:


```
$ bower install bootstrap
bower bootstrap#*      not-cached art://twbs/bootstrap#*
bower bootstrap#*      resolve art://twbs/bootstrap#*
bower bootstrap#*      extract archive.tar.gz
bower bootstrap#*      resolved
art://twbs/bootstrap#e-tag:0b9cb774e1
```

Working with Artifactory without Anonymous Access

By default, Artifactory allows anonymous access to Bower repositories. This is defined under **Security | General Configuration**. For details please refer to [Allow Anonymous Access](#).

If you want to be able to trace how users interact with your repositories you need to uncheck the [Allow Anonymous Access](#) setting. This means that users will be required to enter their username and password.

Unfortunately, the Bower command line tool does not support authentication and you will need to add your credentials to the URL of the bower registry configured in `~/ .bowerrc`:

Replacing the default registry with credentials

```
{
  "registry":
  "http://admin:password@localhost:8081/artifactory/api/bower/bower-repo"
}
```

Use an encrypted password

Use an encrypted password instead of clear-text; see [Centrally Secure Passwords](#).

Cleaning Up the Local Bower Cache

The Bower client saves caches of packages that were downloaded, as well as metadata responses.

We recommend removing the Bower caches (both packages and metadata responses) before using Artifactory for the first time. This is to ensure that your caches only contain elements that are due to requests from Artifactory and not directly from <http://bower.herokuapp.com>.

To clear the bower cache use:

Clean Bower Cache

```
bower cache clean
```

Automatically Rewriting External Dependencies

Packages requested by the Bower client frequently use external dependencies as defined in the packages' `bower.json` file. These dependencies may, in turn, need additional dependencies. Therefore, when download an Bower package, you may not have full visibility into the full set of dependencies that your original package needs (whether directly or transitively). As a result, you are at risk of downloading malicious dependencies from unknown external resources. To manage this risk, and maintain the best practice of consuming external packages through Artifactory, you may specify a "safe" whitelist from which dependencies may be downloaded, cached in Artifactory and configure to rewrite the dependencies so that the Bower client accesses dependencies through a virtual repository as follows:

- Check **Enable Dependency Rewrite** in the Bower virtual repository [advanced configuration](#).
- Specify a whitelist patterns of external resources from which dependencies may be downloaded.
- Specify the remote repository in which those dependencies should be cached.

It is preferable to configure a dedicated remote repository for that purpose so it is easier to maintain.

In the example below the external dependencies will be cached in "bower" remote repository and only package from <https://github.com/jfrogdev> are allowed to be cached.

New Virtual Repository

Basic | **Advanced**

Artifactory Requests Can Retrieve Remote Artifacts ?

External Dependency Rewrite

Enable Dependency Rewrite

Remote Repository For Cache

bower

Patterns Whitelist ?

New Pattern

/github.com/jfrogdev/

Rewriting Workflow

1. When downloading a Bower package, Artifactory analyzes the list of dependencies needed by the package.
2. If any of the dependencies are hosted on external resources (e.g. on github.com), and those resources are specified in the white list,
 - a. Artifactory will download the dependency from the external resource.
 - b. Artifactory will cache the dependency in the remote repository configured to cache the external dependency.
 - c. Artifactory will then modify the dependency's entry in the package's `package.json` file indicating its new location in the Artifactory remote repository cache before returning it to the Bower client.
3. Consequently, every time the Bower client needs to access the dependency, it will be provisioned from its new location in the Artifactory remote repository cache.

Using the Bower Shorthand Resolver

When running `bower install` on a `bower.json` file that is hosted on your local machine, you need to define a custom template in `.bowerrc` file by adding the following line.

```
shorthand-resolver": "art://{{owner}}/{{package}}"
```

From version v4.11, for bower packages downloaded from remote repositories, Artifactory supports resolving dependencies that are specified using the **Bower shorthand resolver for dependencies hosted on GitHub**. Use of the shorthand resolver is reflected in the Bower install output, in the shorthand resolver dependencies, which are prefixed with `$$$art-shorthand-resolver$$$`. For example:

```
bower
mypackagetest#$$$art-shorthand-resolver$$$-<username>-mypackagetest-master
.tar.gz
not-cachedart://<username>/mypackagetest#$$$art-shorthand-resolver$$$-<use
rname>-mypackagetest-master.tar.gz
bower
mypackagetest#$$$art-shorthand-resolver$$$-<username>-mypackagetest-master
.tar.gz
resolveart://<username>/mypackagetest#$$$art-shorthand-resolver$$$-<userna
me>-mypackagetest-master.tar.gz
bower
mypackagetest#$$$art-shorthand-resolver$$$-<username>-mypackagetest-master
.tar.gz
resolvedart://<username>/mypackagetest#$$$art-shorthand-resolver$$$-<usern
ame>-mypackagetest-master.tar.gz
```

Registering Bower Packages

From version 4.6, Artifactory is a Bower registry and lets you register bower packages through remote and virtual repositories. This means you can retrieve bower packages directly from your private Git repositories.

When creating private remote repositories, the Registry URL is redundant and can be left as is.

For example, a private Stash server hosted at <http://stash.mycompany.com:7990> with a project named "artifactory" will be registered as follows:

```
bower register artifactory
ssh://git@stash.mycompany.com:7999/artifactory/artifactory.git
```

Once the server is registered, to download a Bower package from the stash server and cache it in the remote Bower repository in Artifactory (ready for access by users) you can simply run

```
bower install artifactory
```

Viewing Individual Bower Package Information

Artifactory lets you view selected metadata of a Bower package directly from the UI.

In the **Artifacts** tab, select **Tree Browser** and drill down to select the *zip/tar.gz* file you want to inspect. The metadata is displayed in the **Bower Info** tab.

Package Info

Name: bootstrap
Description: The most popular front-end framework for developing responsive, mobile first projects on the web.
License: MIT
Keywords: "css", "js", "less", "mobile-first", "responsive", "front-end", "framework", "web"

Dependencies

< page 1 of 1 >

Name	Version
jquery	>= 1.9.1

Main Files

less/bootstrap.less
dist/js/bootstrap.js

Ignored Files

/*
_config.yml
CNAME

Chef Cookbook Repositories

Overview

Artifactory supports [Chef Cookbook](#) repositories on top of its [existing support](#) for advanced artifact management.

Artifactory support for Chef Cookbook provides:

1. The ability to provision Cookbook packages from Artifactory to the Knife and Berkshelf command line tool from all repository types.
2. Calculation of metadata for Cookbook packages hosted in Artifactory local repositories.
3. Access to remote Cookbook repositories (in particular the [Chef supermarket](#) public repository) through [remote repositories](#) which provide proxy and caching functionality.
4. The ability to access multiple Cookbook repositories from a single URL by aggregating them under a Virtual Repository. This overcomes the limitation of the Knife client which

- can only access a single repository at a time.
5. Compatibility with the Knife command line tool to list, show and install Cookbooks.
 - Compatibility with the Berkshelf command line to resolve Cookbook dependencies.
 6. The ability to assign access privileges according to projects or development teams.

Chef Repository

Chef uses the concept of a [Chef repository](#), to represent storing their own data objects on a workstation. This is different from the use of "repository" in Artifactory.

Chef provides an official "supermarket" for cookbook packages, so Chef repositories in Artifactory are actually Chef supermarkets in Chef terminology. This page refers to Chef Cookbook repositories and Chef supermarkets interchangeably.

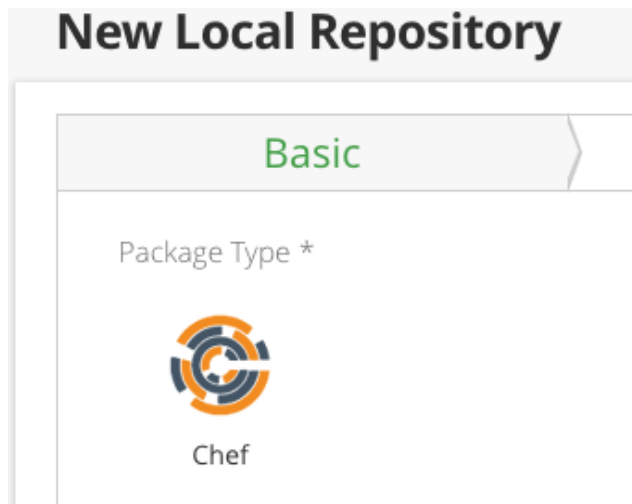
Page contents

- Overview
- Configuration
 - Local Chef Supermarket
 - Repository Layout
 - Remote Chef Supermarket
 - Virtual Chef Supermarket
- Using the Knife Command Line
- Working with Artifactory without Anonymous Access
- Publishing Cookbooks
- Using the Berkshelf Command Line
- Viewing Individual Chef Cookbook Information
- Searching Chef Cookbooks

Configuration

Local Chef Supermarket

To enable calculation of Chef package metadata in local repositories so they are, in effect, Chef supermarkets, set the **Package Type** to **Chef** when you create the repository:



Repository Layout

Artifactory allows you to define any layout for your Chef Cookbook repositories. In order to upload packages according to your custom layout,

you need to package your Chef Cookbook files with Knife or Berkshelf and archive the files as **tar.gz**. Then you can upload to any path within your local Chef supermarket, see [publishing Cookbooks](#).

Remote Chef Supermarket

A **Remote Repository** defined in Artifactory serves as a caching proxy for a supermarket managed at a remote URL such as <https://supermarket.chef.io>.

Artifacts (such as **tgz** files) requested from a remote repository are cached on demand. You can remove downloaded artifacts from the remote repository cache, however, you can not manually deploy artifacts to a remote Chef repository.

To define a remote repository to proxy a remote Chef Cookbook, repository follow the steps below:

1. In the **Admin** module, under **Repositories | Remote**, click "New".
2. In the New Repository dialog, set the **Package Type** to **Chef**, set the **Repository Key** value, and specify the URL to the remote repository in the **URL** field as displayed below.

The screenshot shows a 'New Remote Repository' dialog with three tabs: 'Basic', 'Advanced', and 'Replications'. The 'Basic' tab is selected. Under 'Package Type *', the 'Chef' option is chosen, represented by the Chef logo. Below this, the 'Repository Key *' field contains 'chef-remote' and the 'URL *' field contains 'https://supermarket.chef.io'. A 'Test' button is located to the right of the URL field.

3. Click "Save & Finish".

Virtual Chef Supermarket

A Virtual Repository defined in Artifactory aggregates packages from both local and remote repositories.

This allows you to access both locally hosted Chef Cookbook packages and remote proxied Chef Cookbook repositories from a single URL defined for the virtual repository.

To define a virtual Chef Cookbook repository, create a **virtual repository**, set the **Package Type** to be **Chef**, and select the underlying local and remote Chef repositories to include in the **Basic** settings tab.

Repositories

Available Repositories

- chef-repo
- chef-supermarket
- chef-virtual2

Selected Repositories

- chef-local
- chef-remote

Navigation arrows: <<, <, >, >>

Included Repositories

When configuring the order of resolution note that Artifactory will always resolve first from local repositories, then cache and only then will try to request artifacts from remote repository.

- chef-local
- chef-remote

Using the Knife Command Line

Chef repositories must be prefixed with `api/chef` in the path

When accessing a Chef supermarket through Artifactory, the repository URL must be prefixed with `api/chef` in the path. This applies to all Knife commands.

For example, if you are using Artifactory standalone or as a local service, you would access your Chef supermarket using the following URL:

```
http://localhost:8081/artifactory/api/chef/<repository key>
```

Or, if you are using Artifactory SaaS the URL would be:

```
https://<server name>.jfrog.io/<server name>/api/chef/<repository key>
```

To use the Knife command line you need to make sure it's installed. It's part of ChefDK, that can be [installed in various ways](#).

Once you have created your Chef supermarket, you can select it in the Tree Browser and click **Set Me Up** to get code snippets you can use to change your Chef supermarket URL, and deploy and resolve packages using the knife command line tool.

Set Me Up



General

In order to configure your Knife client to work with Artifactory, you need to edit its *knife.rb* file (which can usually be found under `<user-home-dir>/chef/`) and add a reference to your Artifactory Chef repository as a "supermarket_site". For example:

```
1 knife[:supermarket_site] = 'http://localhost:8081/artifactory/api/chef/chef-local'
```

To support authentication which may be required by Artifactory, you need to install the *knife-art* plugin. For installation instructions, please refer to the [Artifactory User Guide](#)). Once the plugin is installed, you can specify your credentials at the beginning of the url as shown below:

```
1 http://admin:AP78qTg9sZidVod8Tfbeqf9QXF@localhost:8081/artifactory/api/chef/chef-local
```

Deploy

To deploy a cookbook using Knife, run:

```
1 knife artifactory share <cookbook-name> [CATEGORY]
```

Resolve

To install a cookbook using Knife, use the below command:

```
1 knife artifactory install <cookbook-name> [VERSION]
```

Set the default Chef supermarket with a URL pointing to a Chef supermarket in Artifactory by editing your `~/.chef/knife.rb` configuration file (the example below uses a repository with the key `chef-virtual`):

Setting the default Chef supermarket for Knife

```
knife[:supermarket_site] =  
'http://localhost:8081/artifactory/api/chef/chef-virtual'
```

knife.rb file location

The `knife.rb` file doesn't exist by default. It can be created with the `knife configure` command. Refer to the [knife documentation](#) for possible `knife.rb` locations.

The location of this file can be overridden with the `--config` parameter when running a knife command

Working with Artifactory without Anonymous Access

By default, Artifactory allows Anonymous Access for Chef repositories. This is defined under **Security | General Configuration**. For details please refer to [Allow Anonymous Access](#).

If you want to be able to trace how users interact with your repositories you need to uncheck the [Allow Anonymous Access](#) setting. This means that users will be required to enter their username and password.

The Knife command line tool does not support basic authentication (it only supports authentication with RSA keys). To enable basic authentication, you will need to install the **knife-art.gem** plugin.

Install knife-art plugin

```
chef gem install knife-art
```

If properly installed you should see the following specific Artifactory commands:

Knife Artifactory plugin commands

```
** ARTIFACTORY COMMANDS **
knife artifactory download COOKBOOK [VERSION] (options)
knife artifactory install COOKBOOK [VERSION] (options)
knife artifactory list (options)
knife artifactory search QUERY (options)
knife artifactory share COOKBOOK [CATEGORY] (options)
knife artifactory show COOKBOOK [VERSION] (options)
knife artifactory unshare COOKBOOK VERSION
```

These commands are a wrapper around the standard Knife supermarket commands, that enable basic authentication. To add these credentials, pre-pend them to the URL of the Chef supermarket configured in your **knife.rb** file:

Setting the default Chef supermarket for Knife with credentials

```
knife[:supermarket_site] =
  'http://admin:password@localhost:8081/artifactory/api/chef/chef-virtual'
```

Use an encrypted password

Use an encrypted password instead of clear-text; see [Centrally Secure Passwords](#).

Publishing Cookbooks

You can use the UI or a simple REST API call to upload the **tgz/tar.gz** containing the Cookbook to a Chef repository.

Artifactory will automatically extract the relevant information from the `metadata.json` to later serve the index and respond properly to client calls. This `metadata.json` file is mandatory. If it does not exist in your cookbook, you can use a Knife command to generate it and then publish it to Artifactory. For example:

Publishing a new Cookbook

```
$ chef generate cookbook myapp
$ knife artifactory share myapp tool
```

Using the Berkshelf Command Line

Currently, using Berkshelf with Artifactory only supports Anonymous access. A plugin to enable authenticated access with Berkshelf will be provided in a forthcoming release of Artifactory.

Berkshelf is a dependency manager for Chef Cookbooks, and is a part of the **ChefDK**.

To resolve dependencies from a Chef supermarket in Artifactory, set the default supermarket in your Berkshelf's Cookbook:

Setting the default Chef supermarket for Berkshelf

```
source 'http://localhost:8081/artifactory/api/chef/chef-virtual'
```

Then you can execute the `berks` command to download the required dependencies from Artifactory:

Resolving dependencies with Berkshelf

```
vagrant@default-ubuntu-1404:~/chef-zero/mycookbook$ berks
Resolving cookbook dependencies...
Fetching 'mycookbook' from source at .
Fetching cookbook index from
http://localhost:8081/artifactory/api/chef/chef-virtual...
Installing apt (5.0.0) from
http://localhost:8081/artifactory/api/chef/chef-virtual ([opscod]
http://localhost:8081/artifactory/api/chef/chef-virtual/api/v1)
Installing chef-apt-docker (1.0.0) from
http://localhost:8081/artifactory/api/chef/chef-virtual ([opscod]
http://localhost:8081/artifactory/api/chef/chef-virtual/api/v1)
Installing chef-yum-docker (1.0.1) from
http://localhost:8081/artifactory/api/chef/chef-virtual ([opscod]
http://localhost:8081/artifactory/api/chef/chef-virtual/api/v1)
Installing compat_resource (12.16.2) from
http://localhost:8081/artifactory/api/chef/chef-virtual ([opscod]
http://localhost:8081/artifactory/api/chef/chef-virtual/api/v1)
Using mycookbook (0.1.0) from source at .
Installing yum (4.1.0) from
http://localhost:8081/artifactory/api/chef/chef-virtual ([opscod]
http://localhost:8081/artifactory/api/chef/chef-virtual/api/v1)
```

Viewing Individual Chef Cookbook Information

Artifactory lets you view selected metadata of a Chef Cookbook directly from the UI.

In the **Artifacts** tab, select **Tree Browser** and drill down to select the `tgz/tar.gz` file you want to inspect. The metadata is displayed in the **Chef Info** tab.

chef-demo-0.5.0.tar.gz Download Actions

General **Chef Info** Effective Permissions Properties Watchers Builds

Package Info

Name: chefdemo
 Version: 0.5.0
 Maintainer: YOUR_COMPANY_NAME
 License: All rights reserved

Description

Installs/Configures chefdemo

Dependencies

0

Filter by Name or Version

Name	Version
apt	-> 2.2
bluepill	-> 2.3
build-essential	-> 2.0
ohai	-> 2.0
runit	-> 1.2
yum-epel	-> 0.3

Platforms

0

Filter by Name or Version

Name	Version
apt	-> 2.2
bluepill	-> 2.3

Searching Chef Cookbooks

Artifactory supports a variety of ways to [search for artifacts](#).

Artifactory also supports `knife search [search terms ...]`:

- For local repositories, it will look for the given terms in the name, description and maintainer fields.
- For remote repositories, the search will be done on the local cache, then the search query will be forwarded to the external repository and the results merged before returned to the client.
- For virtual repositories, the search will be done on local repositories and then on remote repositories, the results merged before returning to the client.

Properties

Artifactory annotates each deployed or cached Chef Cookbook package with at least 3 properties: `chef.name`, `chef.version` and `chef.maintainer`. If available, it will also add `chef.dependencies`, `chef.platforms` multi-valued properties.

You can use [Property Search](#) to search for Chef Cookbook according to their name, version, maintainer, dependencies or platforms requirements.

CocoaPods Repositories

Overview

Artifactory supports [CocoaPods](#) repositories on top its [existing support](#) for advanced artifact management.

Artifactory support for CocoaPods provides:

1. The ability to provision CocoaPods packages from Artifactory to the pod command line tool from local and remote repositories.
2. Calculation of Metadata for pods hosted in Artifactory's local repositories.
3. Access to remote CocoaPods Specs repositories (such as <https://github.com/CocoaPods/Specs>) through [Remote Repositories](#) which provide the usual proxy and caching functionality.
4. The ability to assign access privileges according to projects or development teams.

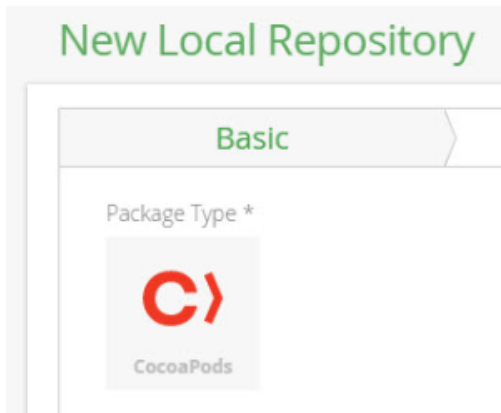
Page Contents

- Overview
- Configuration
 - Local Repositories
 - Deploying Pods
 - Remote Repositories
- Using the Pod Command Line
 - Using cocoapods-art
 - Synchronizing the cocoapods-art Plugin's repositories with Artifactory
- Working with Artifactory without Anonymous Access
- Cleaning Up the Local Pod Cache
- Watch the Screencast

Configuration

Local Repositories

To enable calculation of CocoaPods package metadata set **CocoaPods** to be the **Package Type** when you create your local CocoaPods repository.



Deploying Pods

The CocoaPods client does not provide a way to deploy packages and mostly (though not only) relies on a Git repository to host the pod's code.

To deploy a pod into Artifactory, you need to use Artifactory's [REST API](#) or the [Web UI](#).

A pod is a simple `tar.gz` file which contains your project code as well as a `.podspec` or `.podspec.json` file describing the package metadata.

Pod filetypes

Although more extensions are supported by the client, the Artifactory CocoaPods local repositories currently only support pods that are archived as tar.gz

Remote Repositories

The public [CocoaPods Specs repo](#) does not contain any actual binary packages; it is a git repository containing `podspec.json` files pointing from a package name and version to its storage endpoint.

Since the majority of the packages are hosted on GitHub, you need to create a [Remote Repository](#) which serves as a caching proxy for [github.com](#). If necessary, you can do the same for [bitbucket.org](#) or any other remote repository you want to access.

Working with Stash?

If your packages are hosted on Bitbucket (formerly Stash), you need to ensure that the Stash Archive Plugin is installed on your Bitbucket server.

Artifacts (such as tar.gz files) requested from a remote repository are cached on demand. You can remove downloaded artifacts from the remote repository cache, however you can not manually deploy artifacts to a remote repository.

To define a remote repository to proxy github.com as well as the public Specs repo follow the steps below:

1. Create a new remote repository and set **CocoaPods** to be its **Package Type**
2. Set the **Repository Key** value, and enter `https://github.com` in the **URL** field as displayed below

The screenshot shows the 'New Remote Repository' configuration page. It has three tabs: 'Basic', 'Advanced', and 'Replications'. The 'Basic' tab is active. Under 'Package Type *', there is a dropdown menu with a red 'C' icon and the text 'CocoaPods'. Below that, there are two input fields: 'Repository Key *' with the value 'pods-remote' and 'URL *' with the value 'https://github.com/'. To the right of the URL field is a green 'Test' button.

3. In the **CocoaPods Settings** section, select **GitHub** as the **Git Provider**, and leave the default **Registry URL** (<https://github.com/CocoaPods/Specs>).

Finally, click "Save & Finish"

CocoaPods Settings

Git Provider

GitHub

Specs Repo URL

<https://github.com/CocoaPods/Specs>

Specs Repo URL

Usually, you will point the **Specs Repo URL** field at the public Specs repo as displayed above.

However, if you are using a private Specs repo - set the URL to be the same as the one configured in the **URL** field.

If the remote URL is an Artifactory instance you need to append its url with `/api/pods/<repo>` i.e. `http://art-prod.company.com/artifactory/api/pods/pods-local`

Private Bitbucket server

Working with a private Bitbucket server? set the "Git Provider" to be Stash, and **not** Bitbucket. The public Bitbucket endpoint answers some API calls that a private Bitbucket server doesn't, hence, it is important to set the Git Provider to be Stash when working with a private Bitbucket server. In this case, the URL field should be the root of your Bitbucket server, and the Specs Repo URL should be the full URL to the Specs repo in Bitbucket.

Using the Pod Command Line

CocoaPods repositories must be prefixed with `api/pods` in the path

When accessing a CocoaPods repository through Artifactory, the repository URL must be prefixed with **api/pods** in the path. This applies to the `pod repo-art add` command.

For example, if you are using Artifactory standalone or as a local service, you would access your CocoaPods repositories using the following URL:

```
http://localhost:8081/artifactory/api/pods/<repository key>
```

Or, if you are using Artifactory SaaS, the URL would be:

```
https://<server name>.jfrog.io/<server name>/api/pods/<repository key>
```

Artifactory has been updated to work seamlessly with the latest version of the CocoaPods client from version 0.39.0

Using `cocoapods-art`

In order to use CocoaPods with Artifactory, you need the `cocoapods-art` plugin which presents Artifactory repositories as Specs repos and pod sources.

You can download the `cocoapods-art` plugin as a Gem, and its sources can be found on [GitHub](#).

To use Artifactory with CocoaPods, execute the following steps:

1. Install the `cocoapods-art` plugin:

```
gem install cocoapods-art
```

Using Homebrew?

We recommend installing both the CocoaPods client and the `cocoapods-art` plugin as gems. Installing with Homebrew may cause issues with the CocoaPods hooks mechanism that the plugin relies on.

2. The next step is to add an Artifactory repository by using the `pod 'repo-art add'` command:

```
pod repo-art add <local_specs_repo_name>  
http://localhost:8081/artifactory/api/pods/<repository_key>
```

3. Once the repository is added, add the following in your *Podfile*:

Adding an Artifactory source in the Podfile

```
plugin 'cocoapods-art', :sources => [  
  '<local_specs_repo_name>'  
]
```

Working without the Master repository?

If you have removed the CocoaPods Master repository from your system, due to a known [issue](#) with the CocoaPods stats plugin, you need to add the following to your Podfile, or add the corresponding variable to your environment:

```
ENV['COCOPODS_DISABLE_STATS'] = 'true'
```

For details, please refer to [JFrog Jira](#)

Where the local repo name is the name you gave the specs repo locally when adding it.

pod repo-art commands

The *cocoapods-art* plugin exposes most commands that are normally invoked with *pod repo* (i.e. add, update, list etc.). Use **pod repo-art** instead of *pod repo* whenever dealing with Artifactory-backed Specs repositories.

CocoaPods local Specs repos location

`~/cocoapods/repos`

Once the pod command line tool is configured, every `pod install` command will fetch pods from the CocoaPods repository specified above.

Synchronizing the cocoapods-art Plugin's repositories with Artifactory

As opposed to the cocoapods client's default behavior, the cocoapods-art plugin does not automatically update its index whenever you run client commands (such as install). To keep your plugin's index synchronized with your CocoaPods repository, you need to update it by executing the following command:

```
pod repo-art update
```

Working with Artifactory without Anonymous Access

By default, Artifactory allows anonymous access to CocoaPods repositories. This is defined under **Security | General Configuration**. For details please refer to [Allow Anonymous Access](#).

If you want to be able to trace how users interact with your repositories you need to uncheck the [Allow Anonymous Access](#) setting. This means that users will be required to enter their username and password.

Unfortunately, the pod command line tool does not support authentication against http endpoints. The *cocoapods-art* plugin solves this by forcing curl (which pod uses for all http requests) to use the `.netrc` file:

.netrc file example

```
machine art-prod.company.com
login admin
password password
```

Since Artifactory also supports basic authentication using your [API key](#), you could use that instead of your password:

.netrc file using your API key

```
machine art-prod.company.com
login admin
password
AKCp2TfQM58F8FTkXo8qSJ8NymwJivmgefBqoJeEBQLSHCZusEH6Z2dmhS1siSxZTHoPPyUW
```

Use an encrypted password

We recommend using an encrypted password instead of clear-text. For details, please refer to [Centrally Secure Passwords](#).

Cleaning Up the Local Pod Cache

The pod client saves caches of pods that were downloaded, as well as metadata.

We recommend removing the CocoaPods caches (both packages and metadata responses) before using Artifactory for the first time. This is to ensure that your caches only contain elements that are due to requests from Artifactory and not directly from other Specs repos.

To clear the pod cache use:

```
pod cache clean
```

Watch the Screencast

Watch this short screencast to learn how easy it is to host RPMs in Artifactory.

Conan Repositories

Overview

Artifactory introduces advanced artifact management to the world of C/C++ through support for local repositories that work directly with the **Conan** client to manage Conan packages and dependencies. As a repository to which builds can be uploaded, and from which dependencies can be downloaded, Artifactory offers many benefits to C/C++ developers using Conan:

1. Secure, private repositories for C/C++ packages with fine-grained access control according to projects or development teams
2. Automatic layout and storage of C/C++ packages for all platforms configured in the Conan client
3. The ability to provision C/C++ dependencies from Artifactory to the Conan command line tool from local repositories.
4. Enterprise features such as high availability, repository replication for multi-site development, different options for massively scalable storage

...and much more.

For more details on building Conan packages and working with the Conan client, please refer to the **Conan documentation**.

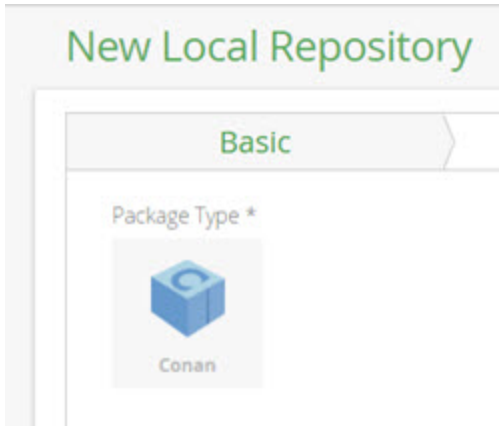
Configuration

Local Repositories

To enable calculation of C/C++ package metadata, set **Conan** to be the **Package Type** when you create your local repository.

Page Contents

- Overview
- Configuration
 - Local Repositories
- Using Conan with Artifactory
 - Adding Your Repository
 - Authenticating the Conan Client
 - Installing Dependencies
 - Uploading Packages
- Viewing Individual Conan Package Information



Make sure to also select `conan-default` as the repository layout.

Using Conan with Artifactory

Once the Conan client is installed, you can access Conan repositories in Artifactory through its command line interface. You can only install packages from or export packages to your Artifactory local Conan repository using the Conan client.

Local vs. Remote

Don't let Conan terminology confuse you. For the purposes of this integration, the Conan "Remote" is actually the Artifactory local repository you have created for Conan packages.

Once you have created your Conan repository, select it in the Tree Browser and click **Set Me Up**, to see the code snippets you will need in order to use your repository as a source to install packages and as a target for export.

Set Me Up ×

Tool Remove User Data

Conan

Repository

conan-local

General

To add the repository to your conan cli, use:

```
1 conan remote add <REMOTE> http://[redacted]:[redacted]/artifactory/api/conan/conan-local
```

And replace <REMOTE> with a name that identifies the repository (for example: "my-conan-repo")

To login use the *conan user* command:

```
1 conan user -p Al [redacted]@G -r <REMOTE> admin
```

And provide your Artifactory username and password or API key.
If anonymous access is enabled you do not need to login.
For complete conan cli reference see documentation at docs.conan.io.

In the sections below, <REMOTE> is used to denote the logical name you set with which the Conan client can identify the Conan local

repository in Artifactory.

Adding Your Repository

To use your local repository with Conan, you first need to add it as a Conan "Remote" to the client as follows:

```
conan remote add <REMOTE> http://<ARTIFACTORY_URL>/api/conan/<REPO_KEY>
```

Where:

<REPO_KEY> is the repository key.

Conan repositories must be prefixed with `api/conan` in the path

When accessing a Conan repository through Artifactory, the repository URL must be prefixed with `api/conan` in the path. This applies to all Conan commands including `conan install`.

For example, if you are using Artifactory standalone or as a local service, you would access your Conan repositories using the following URL:

```
http://localhost:8081/artifactory/api/conan/<repository key>
```

Or, if you are using Artifactory SaaS, the URL would be:

```
https://<server name>.jfrog.io/<server name>/api/conan/<repository key>
```

Authenticating the Conan Client

To authenticate the Conan client to Artifactory you need to log in using:

```
conan user -p <PASSWORD> -r <REMOTE> <USERNAME>
```

Accessing Artifactory anonymously

If Artifactory is configured for [anonymous access](#), you may skip authenticating the Conan client.

Installing Dependencies

To install dependencies from Artifactory as defined in your `conanfile.txt` file use:

```
conan install . -r <REMOTE>
```

Uploading Packages

To upload packages to your Artifactory local Conan repository use:

```
conan upload <RECIPE> -r <REMOTE> --all
```

Where <RECIPE> specifies your Conan recipe reference formatted <NAME>/<VERSION>@<USER>/<CHANNEL>

Viewing Individual Conan Package Information

Artifactory lets you view selected metadata of a Conan package directly from the UI.

In the **Artifacts** tab, select **Tree Browser** and drill down to select the package file you want to inspect. The metadata is displayed in the **Conan Info** tab. The specific information displayed depends on the tree item you have selected. Selecting the root item of a package displays details of the Conan recipe used to upload the package.

Artifact Repository Browser

Tree Simple Q Compress Empty Folders

Poco/1.7.3/andy_somerville/stable

General Conan Info Effective Permissi... Properties Watchers

Recipe Info

Name:	Poco
Version:	1.7.3
User:	andy_somerville
Channel:	stable
Reference:	Poco/1.7.3@andy_somerville/stable
URL:	http://github.com/lasote/conan-poco

Packages Info

Package Count:	2
----------------	---

If you select one of the packages, you get detailed Conan Package info including **Settings**, **Options** and dependencies ("**Requires**")

Tree Simple Q Compress Empty Folders

aa56febbace1eff7e4271176442fbacdee655e8

General Conan Package Info Effective Permissi... Properties Watchers

Settings

OS:	Linux
Architecture:	x86_64
Build Type:	Release
Compiler:	gcc
Compiler Version:	5.4
compiler.libcxx:	libstdc++

Options

enable_tests:	False
Shared:	False
enable_xml:	True
enable_data_odbc:	False
enable_sevenzzip:	False
enable_pocodoc:	False
force_openssl:	True
enable_data_sqlite:	True
poco_unbundled:	False
enable_pdf:	False
enable_json:	True
enable_zip:	True
enable_net:	True
cxx_14:	False
enable_netssl_win:	True

Requires

- OpenSSL/1.0.2h@lasote/stable:3e9773e910c426c97be68f9d0ad2926754705ba9
- electric-fence/2.2.0@lasote/stable:500a5737cfb7bee13d4a0039e72446892ca242ab
- zlib/1.2.8@lasote/stable:500a5737cfb7bee13d4a0039e72446892ca242ab

Debian Repositories

Overview

From version 3.3, Artifactory supports Debian repositories whether they use the **current Automatic Debian** architecture or the deprecated **Trivial** architecture. As a fully-fledged Debian repository, Artifactory generates index files that are fully compliant with Debian clients.

Artifactory support for Debian provides:

1. The ability to provision Debian packages from Artifactory to a Debian client from local and remote repositories.
2. Calculation of Metadata for Debian packages hosted in Artifactory's local repositories.
3. Access to remote Debian resources (such as *us.archive.ubuntu.com*) through **Remote Repositories** which provide the usual proxy and caching functionality.
4. Providing GPG signatures that can be used by Debian clients to verify packages.
5. Complete management of GPG signatures using the Artifactory UI and the REST API.

Configuration

You can only deploy Debian packages to a local repository that has been created with the Debian **Package Type**.

You can download packages from a local or a remote Debian repository.

Page Contents

- Overview
- Configuration
 - Local Repositories
 - Deploying a package using the UI
 - Deploying a package using Matrix Parameters
 - Setting the Target Path
 - Specifying multiple layouts
 - Artifact Metadata
 - Metadata Validation
 - Remote Repositories
- Signing Debian Packages
- Adding MD5 Checksum to the Packages file
- Authenticated Access to Servers
- Compression Formats
- Acquiring Packages by Hash
- REST API Support
- Watch the Screencast

Local Repositories


To create a new local repository that supports Debian, under the **Basic** settings, set the **Package Type** to be **Debian**.

If you are using Debian with a *Trivial* layout, in the **Debian Settings** section, set the **Trivial Layout** checkbox.

New Local Repository

Basic > Advanced > Replications

Package Type *

 **debian**
Debian

Repository Key *

General

Repository Layout

simple-default

Debian Settings

Trivial Layout

Deploying a package using the UI

To deploy a Debian package to Artifactory, in the Artifactory Repository Browser, click **Deploy**.

Select your Debian repository as the **Target Repository**, upload the file you want to deploy.


Deploy

Target Repository

debian-local

Package Type: Debian

Type: **Single** Multi



Uploading planckdb-08-2015.deb

Target Path

pool/planckdb-08-2015.deb

Check the **Deploy as Debian Artifact** checkbox and fill in the **Distribution**, **Component** and **Architecture** fields in the **Debian Artifact** section. Notice that the **Target Path** is automatically updated to reflect your input.

Deploy ×

Type: **Single** Multi

Uploading planckdb-08-2015.deb

✕

Target Path

Deploy as Debian Artifact

Debian Artifact

Distribution	Component
<input type="text" value="trusty"/>	<input type="text" value="main"/>
Architecture	
<input type="text" value="amd64,i386"/>	

Deploy

Setting the target path manually? Be careful with spaces

We recommend using the fields in the **Debian Artifact** section to set your **Target Path**. Nevertheless, if you choose to specify the **Target Path** manually, make sure you don't enter any superfluous spaces.

For example to upload package **planckdb-08-2015.deb**, and specify that its layout is from the **trusty** distribution, in the **main** component and the **i386** architecture, you would enter:

```
pool/planckdb-08-2015.deb;deb.distribution=trusty;deb.component=main;deb.architecture=i386
```

You can also deploy Debian packages to Artifactory with an explicit URL using [Matrix Parameters](#).

After you deploy the artifact, you need to wait about one minute for Artifactory to recalculate the repository index and display your upload in the Repository Browser.

Once you have deployed your Debian package, and Artifactory has recalculated the repository index, your repository should be organized as displayed below:

The screenshot shows the 'Artifact Repository Browser' interface. On the left is a tree view of the repository structure, with 'planckdb-08-2015.deb' selected under the 'pool' directory. The main panel displays the details for this package, including its name, repository path, size, creation and modification dates, and checksums.

Info	
Name:	planckdb-08-2015.deb
Repository Path:	debian-local/pool/planckdb-08-2015.deb
Module ID:	N/A
Deployed by:	admin
Size:	75.32 KB
Created:	01-10-14 09:56:41 UTC
Last Modified:	01-10-14 09:55:37 UTC
Licenses:	Not Found Query Code Center
Downloaded:	1
Last Downloaded:	23-07-15 13:17:41 UTC
Last Downloaded By:	._system_
<input type="checkbox"/> Filtered	

Checksums	
SHA-1:	c8d852a73e15c4f877a18f756c807ab2b295e (Uploaded: Identical)
MDS:	692315a711ffda1e672693e076182555 (Uploaded: Identical)

Deploying a package using Matrix Parameters

The URL is built similarly to the [Target Path](#) format as follows:

Deploying a package using Matrix Parameters

```
PUT
"http://$ARTIFACTORY_HOME/{debianRepoKey}/pool/{debianPackageName};deb.distribution={distribution};deb.component={component};deb.architecture={architecture}"
```

For example, to upload package **libatk1.0_i386.deb**, and specify that its layout is from the **wheezy** distribution, in the **main** component and the **i386** architecture, you would enter:

Example

```
PUT
"http://localhost:8080/artifactory/debian-local/pool/libatk1.0_i386.deb;deb.distribution=wheezy;deb.component=main;deb.architecture=i386"
```

Setting the Target Path

The **Target Path** needs to be entered in a strict and specific format that uses system properties to define where the artifact will be stored and its specific layout as follows:

Target Path Format

```
[path];deb.distribution=[distribution];deb.component=[component];deb.architecture=[architecture]
```

path	The repository path where the package should be stored. Artifactory supports storing Debian packages anywhere within the repository. The examples on this page show Debian packages stored under the pool folder in accordance with the Debian convention.
distribution	The value to assign to the <code>deb.distribution</code> property used to specify the Debian package distribution
component	The value to assign to the <code>deb.component</code> property used to specify the Debian package component name
architecture	The value to assign to the <code>deb.architecture</code> property used to specify the Debian package architecture

Adding Architecture Independent Packages

Uploading a Debian package with `deb.architecture=all` will cause it to appear in the Packages index of all the other architectures under the same Distribution and Component, as well as under a new index branch called `binary-all` which holds all Debian packages that are marked as "all".

Removing an "all" Debian package will also remove it from all other indexes under the same Distribution and Component.

When the last Debian package in an architecture is removed but the Packages index still contains an "all" Debian package, it is preserved in the index.

If you want such an architecture index removed you may do so via the UI or using [Calculate Debian Repository Metadata](#) in the REST API, which cleans up orphaned package files from the index.

Specifying multiple layouts

Whether uploading a package using the UI or Matrix Parameters, you can specify multiple layouts for any Debian package you upload, by including additional values for the distribution, component or architecture separated by a comma,

For example, to upload package **libatk1.0_i386.deb** to both **wheezy** and **trusty** distributions, in both **main** and **contrib** components and both **i386** and **64bit-arm** architectures you would specify the following Target Path to upload using the UI:

Target path for multiple layouts

```
pool/libatk1.0_i386.deb;deb.distribution=wheezy;deb.distribution=trusty;deb.component=main;deb.component=contrib;deb.architecture=i386;deb.architecture=64bit-arm
```

Correspondingly, to upload the file using Matrix Parameters, you would use the following:

Multiple layouts using Matrix Parameters

```
PUT
"http://localhost:8080/artifactory/debian-local/pool/libatk1.0_i386.deb;deb.distribution=wheezy;deb.distribution=trusty;deb.component=main;deb.component=contrib;deb.architecture=i386;deb.architecture=64bit-arm"
```

Artifact Metadata

From version 4.4, Artifactory writes several entries from the Debian package's metadata as properties on all of the artifacts (based on the control file's content).

These properties can be used to search for Debian packages more efficiently using Artifactory's [Package Search](#).

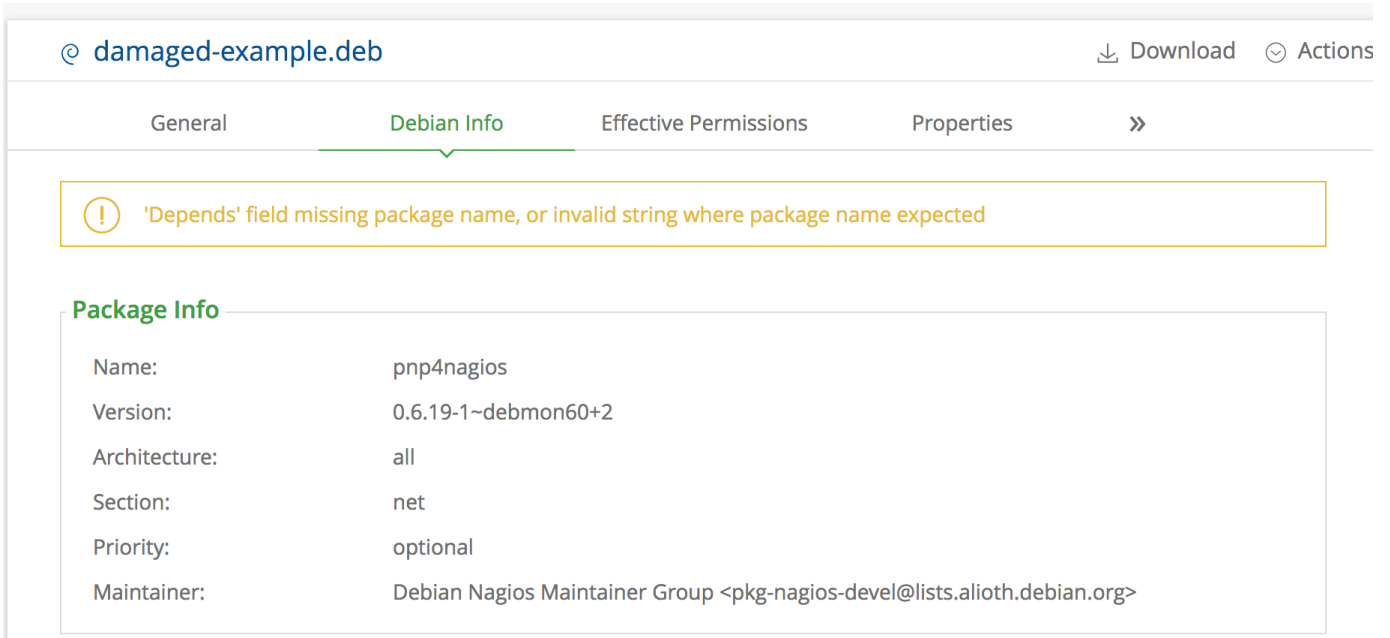
Metadata properties are written for each new Artifact that is deployed to Artifactory.

To have these properties written to Debian artifacts that already exist in your repositories you need to call the [Calculate Debian Repository Metadata REST API](#) which writes the properties to all of the artifacts by default.

Metadata Validation

To ensure that Debian repositories are not corrupted by malformed packages, Artifactory first validates parts of the Debian metadata to make sure that none of the relevant metadata fields are empty. If the validation process indicates a malformed package, Artifactory provides several indications:

- The package is not indexed
- The package is annotated with the following property:
 - key: `deb.index.status`
 - value: the reason the package failed the validation process
- If the package is selected in the Tree Browser, the Debian Info tab will display a message indicating that it was not indexed and why it failed the validation process



The screenshot shows the Artifactory interface for a package named `damaged-example.deb`. The `Debian Info` tab is active, displaying a yellow warning box with an exclamation mark icon and the message: `'Depends' field missing package name, or invalid string where package name expected`. Below the warning, the `Package Info` section lists the following details:

Name:	pnp4nagios
Version:	0.6.19-1~debmon60+2
Architecture:	all
Section:	net
Priority:	optional
Maintainer:	Debian Nagios Maintainer Group <pkg-nagios-devel@lists.alioth.debian.org>

- A message is logged in the Artifactory log file indicating that the package was not indexed and why it failed the validation process.

Disable validation

Debian package validation is controlled by the `debian.metadata.validation` [system property](#). Package validation is enabled by default. To disable Debian package validation set:
`debian.metadata.validation=false`

Finding malformed packages

To easily find all malformed packages in your Debian repositories, you can use [Property Search](#) or run an AQL query with [Properties Criteria](#) on the `deb.index.status` property described above.

Remote Repositories

You can download Debian packages from Local Debian Repositories as described above, or from Remote Repositories specified as supporting Debian packages.

To specify that a Remote Repository supports Debian packages, you need to set its **Package Type** to **Debian** when it is created.

Basic

Package Type *



Note that the index files for remote Debian repositories (including the sources index) are stored and renewed according to the [Retrieval Cache Period](#) setting.

Signing Debian Packages

Artifactory supports signing Debian packages using a GPG key. This process will create a signed Release file named `Release.gpg` that will be shipped alongside the Release file. Artifactory will store and manage public and private keys that are used to sign and verify Debian packages.

To generate a pair of GPG keys and upload them to Artifactory, please refer to [GPG Signing](#).

Adding MD5 Checksum to the Packages file

To support tools (e.g. [Aptly](#)) that require Debian packages to include their MD5 checksum in their `packages` metadata file for validation, you can configure Artifactory to add this value by setting the following system property in the `artifactory.system.properties` file:

```
## Add package MD5 checksum to Debian Packages file
#artifactory.debian.metadata.calculateMd5InPackagesFiles=true
```

Artifactory needs to be restarted for this change to take effect.

Authenticated Access to Servers

If you need to access a secured Artifactory server that requires a username and password, you can specify these in your Debian `source.list` file by prefixing the artifactory host name with the required credentials as follows:

Accessing Artifactory with credentials

```
http://user:password@$ARTIFACTORY_HOME/{repoKey} {distribution}
{components}
For example:
http://admin:password@localhost:8081/artifactory/debian-local wheezy main
restricted
```

Encrypting your password

You can use your encrypted password as described in [Using Your Secure Password](#).

Compression Formats

Artifactory supports the following compression formats for Debian indices:

- Gzip (.gz file extension)
- Bzip2 (.bz2 file extension)

Acquiring Packages by Hash

From version 5.5.2, Artifactory supports the acquire-by-hash functionality of APT clients for Debian repositories laid out using the Automatic architecture (Trivial architecture is not supported for acquiring packages by hash). This feature is supported by two [system properties](#):

debian.use.acquire.byhash	[default: true] When true, the value of acquire-by-hash in Debian release files is set to true allowing APT clients to access Debian packages by their checksums (MD5, SHA1, SHA256). To allow this, Artifactory will add the "by-hash" layout to all Debian repositories
debian.packages.byhash.history.cycles.to.Keep	[default: 3] Specifies the number of cycles of package file history to save when acquire-by-hash is enabled

REST API Support

The Artifactory REST API provides extensive support for Debian signing keys and recalculating the repository index as follows:

- [Set the public key](#)
- [Get the public key](#)
- [Set the private key](#)
- [Set the pass phrase](#)
- [Recalculate the index](#)

Watch the Screencast

Docker Registry

Set up a secure private Docker registry in minutes to manage all your Docker images while exercising fine-grained access control. Artifactory places no limitations and lets you set up any number of Docker registries, through the use of local, remote and virtual Docker repositories, and works transparently with the Docker client to manage all your Docker images, whether created internally or downloaded from remote Docker resources such as Docker Hub.

Multiple Docker Registries

Artifactory lets you define as many Docker registries as you wish. This enables you to manage each project in a distinct registry and exercise better access control to your Docker images.

Use Docker Naturally

Artifactory supports the relevant calls of the [Docker Registry API](#) so that you can transparently use the Docker client to access images through Artifactory.

Secure private Docker Registry with Fine-grained Access Control

[Local Docker repositories](#) are where you store internal Docker images for distribution across your organization. With the fine-grained access control provided by built-in [security features](#), Artifactory offers secure Docker push and pull with local Docker repositories as fully functional, secure, private Docker registries.

Consistent and reliable access to remote images

[Remote Docker repositories](#) in Artifactory proxy external resources such as Docker Hub, or a remote Docker repository in another Artifactory instance, and cache downloaded images. As a result, overall networking is reduced, and access to images on these remote resources is faster, consistent and reliable.

Confidently Promoting Images to Production

Artifactory lets you promote Docker images, as immutable, stable binaries, through the quality gates all the way to production.

Smart Search

Using Artifactory's Package Search, find your images in the most natural way for Docker using the image name, tag or digest.

JFrog End-to-End Platform

Through Artifactory's tight integration with JFrog Bintray, you can manage your Docker images from development, through your pipeline, all the way to distribution.

Registries and Repositories

Both Artifactory and Docker use the term "repository", but each uses it in a different way.

A **Docker repository** is a hosted collection of tagged images that, together, create the file system for a container

A **Docker registry** is a host that stores Docker repositories

An **Artifactory repository** is a hosted collection of Docker repositories, effectively, a Docker registry in every way, and one that you can access transparently with the Docker client.

Since Artifactory places no limitation on the number of repositories you may create, you can manage any number of Docker registries in Artifactory.

Page Contents

- [Registries and Repositories](#)
- [Getting Started With Artifactory as a Docker Registry](#)
- [Configuring Docker Repositories](#)
 - [Local Docker Repositories](#)
 - [Remote Docker Repositories](#)
 - [Docker Repository Path and Domain](#)
 - [Virtual Docker Repositories](#)
 - [Reverse Proxy Settings](#)
- [Promoting Docker Images](#)
- [Pushing and Pulling Images](#)
 - [Set Me Up](#)
- [Browsing Docker Repositories](#)
 - [Tag Info](#)
 - [Docker Tag Visualization](#)
 - [Labels](#)
- [Searching for Docker Images](#)
- [Listing Docker Images](#)
- [Pushing Images to Bintray](#)
- [Deletion and Cleanup](#)
 - [Limiting Unique Tags](#)
- [Migrating from Docker](#)

- [V1 to Docker V2 Support Matrix](#)

Read More

- [Getting Started with Artifactory as a Docker Registry](#)
- [Advanced Topics](#)
- [Working with Docker Content Trust](#)
- [Using Docker V1](#)

Getting Started With Artifactory as a Docker Registry

There are three main ways to get started using Docker with Artifactory:

1. [Artifactory SaaS account](#)
2. [Using Docker Compose \(1-minute setup\)](#)
3. [Artifactory On-Prem](#)

For more details, please refer to [Getting Started with Artifactory as a Docker Registry](#).

Configuring Docker Repositories

Artifactory supports three types of repositories when working with Docker:

Local repositories are a place for your internal Docker images. Through Artifactory's security capabilities, these are secure private Docker registries.

Remote repositories are used to proxy remote Docker resources such as Docker Hub.

Virtual repositories can aggregate multiple Docker registries thus enabling a single endpoint you can use for both pushing and pulling Docker images. This enables the admin to manage the different Docker registries without his users knowing, and continue to work with the same endpoint.

**Make sure to go to the Advanced tab of each repository and set the Registry Port if you are using the Port method for Docker. Then, the reverse proxy generator should add a new section in for the specified port.

Local Docker Repositories

A local Docker repository is where you can deploy and host your internal Docker images. It is, in effect, a Docker registry able to host collections of tagged Docker images which are your Docker Repositories. Once your images are hosted, you can exercise fine-grained access control, and share them across your organization through replication or by being proxied by repositories in other Artifactory instances.

To define a local Docker repository, follow the steps below:

1. Create a new [Local Repository](#) and set **Docker** as the **Package Type**.
2. Set the **Repository Key**, and in the Docker Settings section, select **V2** as the Docker API version.
3. Set **Max Unique Tags**. This specifies the maximum number of unique tags, per repository, that should be stored for a Docker image. Once the number of tags for an image exceeds this number, older tags will be removed. Leaving the field blank (default) means all tags will be stored.

New Local Repository

Basic | Advanced | Replications

Repository Key *
docker-local

General

Repository Layout
simple-default

Public Description

Internal Description

Docker Settings

API Version
 V1 V2

Max Unique Tags ?
5

Remote Docker Repositories

With Docker, you can proxy a remote Docker registry through remote repositories. A **Remote Repository** defined in Artifactory serves as a caching proxy for a registry managed at a remote URL such as `https://registry-1.docker.io/` (which is the Docker Hub), or even a Docker repository managed at a remote site by another instance of Artifactory.


Docker images requested from a remote repository are cached on demand. You can remove downloaded images from the remote repository cache, however, you can not manually push Docker images to a remote Docker repository.

To define a remote repository to proxy a remote Docker registry follow the steps below:

1. Create a new Remote Repository and set **Docker** as the **Package Type**.
2. Set the **Repository Key** value, and specify the URL to the remote registry in the **URL** field

New Remote Repository

Basic | Advanced | Replications

Package Type *
 Docker

Repository Key *
docker-remote1

URL *
https://registry-1.docker.io/ Test

General

Repository Layout
simple-default

Remote Layout Mapping
Select Remote Layout

Public Description

Internal Description

Include Patterns ?
New Pattern

Exclude Patterns ?
New Pattern

Offline ?

Docker Settings

API Version: V2

Enable Token Authentication ?

If you are proxying the Docker Hub, use `https://registry-1.docker.io/` as the URL, and make sure the **Enable Token Authentication** check box is checked (these are the default settings).

Docker Repository Path and Domain

When accessing a remote Docker repository through Artifactory, the repository URL must be prefixed with **api/docker** in the path.

For Example:

`http://my-remote-site:8081/artifactory/api/docker/<repository key>`

Virtual Docker Repositories

From version 4.1, Artifactory supports virtual Docker Repositories. A [Virtual Repository](#) defined in Artifactory aggregates images from both local and remote repositories that are included in the virtual repositories.

This allows you to access images that are hosted locally on local Docker repositories, as well as remote images that are proxied by remote Docker repositories, and access all of them from a single URL defined for the virtual repository. Using virtual repositories can be very useful since users will continue to work with the virtual repository while the admin can manage the included repositories, replace the default deployment target and those changes will be transparent to the users.

To define a virtual Docker repository follow the steps below:

1. Create a new Virtual Repository and set **Docker** as the **Package Type**.
2. Set the **Repository Key** value.
3. Select the underlying local and remote Docker repositories to include under the **Repositories** section.
4. You can optionally also configure your **Default Deployment Repository**. This is the repository to which Docker images uploaded to this virtual repository will be routed, and once this is configured, your virtual Docker repository is a fully-fledged Docker registry. Using the default deployment repository, you can set up your virtual repository to wrap a series of repositories that represent the stages of your pipeline, and then [promote images](#) from the default deployment repository through the pipeline to production. Any repository that represents a stage in your pipeline within this virtual repository can be configured with permissions for authenticated or unauthenticated (anonymous) access according to your needs.

Repositories

Available Repositories

<<

<

>

>>

Selected Repositories

📦 docker-dev-local2✕

📦 docker-prod-local2✕

📦 docker-remote✕

Included Repositories

docker-dev-local2

docker-prod-local2

docker-remote

Default Deployment Repository

docker-dev-local2 ▾

Reverse Proxy Settings

A reverse proxy is required for Docker. In case you are using the Artifactory [Reverse Proxy](#) configuration generator you can configure Docker repository's reverse proxy settings under the **Advanced** settings tab.

For details, please refer to [Docker Reverse Proxy Settings](#).

Promoting Docker Images

Artifactory supports promoting Docker images from one Docker repository in Artifactory to another.

Promoting is useful when you need to move Docker images through different acceptance and testing stages, for example, from a development repository, through the different gateways all the way to production. Instead of rebuilding the image multiple times using promotion will ensure the image you will have in your production environment is the one built by your CI server and passed all the relevant tests.

Promotion can be triggered using the following endpoint with cURL:the following endpoint with cURL:

```
POST api/docker/<repoKey>/v2/promote
{
  "targetRepo" : "<targetRepo>",
  "dockerRepository" : "<dockerRepository>",
  "tag" : "<tag>",
  "targetTag" : "<tag>",
  "copy": <true | false>
}
```

where:

repoKey	Source repository key
targetRepo	The target repository to move or copy
dockerRepository	The docker repository name to promote
tag	An optional tag name to promote, if null - the entire docker repository will be promoted. Default: "latest"
targetTag	The new tag that the image should have after being promoted if you want to
copy	When true, a copy of the image is promoted. When false, the image is moved to the target repository

An example for promoting the docker image *"jfrog/ubuntu"* with all of it's tags from *docker-local* to *docker-prod* using cURL would be:

```
curl -i -uadmin:password -X POST "https://artprod.company.com/v2/promote"
-H "Content-Type: application/json" -d
'{"targetRepo": "docker-prod", "dockerRepository": "jfrog/ubuntu"}'
```

Notice that the above example is executed through your reverse proxy. To go directly through Artifactory, you would execute this command as follows:

```
curl -i -uadmin:password -X POST
"http://localhost:8080/artifactory/api/docker/docker-local/v2/promote" -H
"Content-Type: application/json" -d
'{"targetRepo": "docker-prod", "dockerRepository": "jfrog/ubuntu"}'
```

The following example adds retagging with a specific version of the *"jfrog/ubuntu"* image (4.9.0) being retagged to "latest" as it gets promoted:


```
curl -i -uadmin:password -X POST "https://artprod.company.com/v2/promote"
-H "Content-Type: application/json" -d
'{"targetRepo":"docker-prod","dockerRepository":"jfrog/ubuntu", "tag" :
"4.9.0", "targetTag" : "latest"}'
```

Pushing and Pulling Images

Set Me Up

To get the corresponding `docker push` and `docker pull` commands for any repository, select it in the Tree Browser and click **Set Me Up** button.

Set Me Up

Repository

docker-dev-local2

General

Using Docker with Artifactory requires a reverse proxy such as Nginx or Apache. For more details please visit our [Docker Repositories](#) documentation.

In this example we use **artprod.company.com** to represent the Docker repository in Artifactory.

To login use the `docker login` command.

```
1 docker login artprod.company.com
```

And provide your Artifactory username and password or API key. If anonymous access is enabled you do not need to login.

Deploy

To push an image tag an image using the `docker tag` and then `docker push` command.

```
1 docker tag ubuntu artprod.company.com/<DOCKER_REPOSITORY>:<DOCKER_TAG>
```

```
1 docker push artprod.company.com/<DOCKER_REPOSITORY>:<DOCKER_TAG>
```

Resolve

Browsing Docker Repositories

For general information on how to browse repositories, please refer to [Browsing Artifactory](#).

The **Docker Info** tab presents three sections: [Tag Info](#), [Docker Tag Visualization](#), and [Labels](#).

Tag Info

Presents basic details about the selected tag.

Tag Info




Title: label:latest
Digest: sha256:4015a630ed29b07e3d550226dd76868e38c4be25ffa6b6d7927124551558cec4
Total Size: 1.1 MB
Label Count: 2

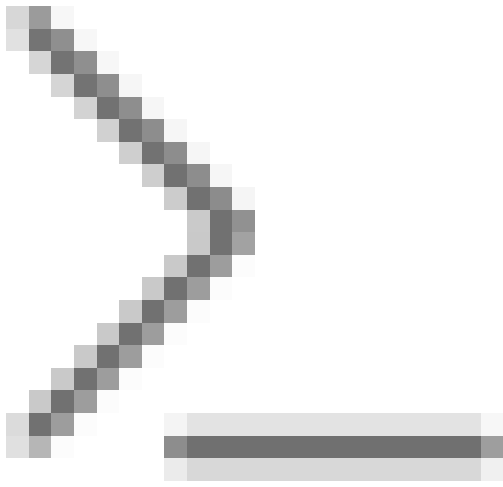
Title	The Docker tag name.
Digest	The tag's SHA 256 digest.
Total Size	The total size of the image
Label Count	The number of labels attached to this tag. <div style="border: 1px solid green; padding: 5px; text-align: center;">Click the label count to view the attached labels at the bottom of the screen.</div>

Docker Tag Visualization

This section maps the entire set of commands used to generate the selected tag along with the digest of the corresponding layer. Essentially, you would see the same series of commands using `docker history`.


You can select any layer of the image to view the following properties:

Symbol	Property
	The layer ID
	The layer size
	The timestamp when the layer was created



The
command
that
created
the layer

Docker tag Visualization

CMD	latest	38000b32c4b5
	38000b32c4b50f9870a033fed7dc5d204736965b93509dfc84b2bfa2a441e...	
	sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c...	
	CMD ["/bin/sh" "-c" "[/bin/sh]"]	
	2016-01-03 16:22:21	 0 B
LABEL		aeebbb1a2a03
CMD		fc0db02f3072
ADD		5c5fb281b01e
	5c5fb281b01ee091a0fffa5b4a4c7fb7d358e7fb7c49c263d6d7a4e35d199f...	
	sha256:d7e8ec85c5abc60edf74bd4b8d68049350127e4102a084f22060f7...	
	ADD file:c295b0748bf05d4527f500b62ff269bfd0037f7515f1375d2ee474b830bad382 in /	
	2015-12-08 18:31:50	 1.1 MB

Labels

This section displays the labels attached to the image.

Labels	
2 Records	
<input type="text" value="Filter by Key"/>	
< Page 1 of 1 >	
Key	Value
artifactory	Example for Label as property
production	Ready for production

Note also, that from version 4.4.0, Artifactory extracts any labels associated with a Docker image and creates corresponding properties on the `manifest.json` file which you can use to specify search parameters, this can be used to easily add additional metadata to any image.

manifest.json Download View Actions

General Effective Permissions **Properties** Watchers Builds Governance

Add: **Property** | Property Set

Name * Value Add

Recursive ?

5 Properties

Filter by Property Delete Page 1 of 1

Property	Value(s)
sha256	4015a630ed29b07e3d550226dd76868e38c4be25ffa6b6d7927124551558cec4
docker.manifest	latest
docker.repoName	label
docker.label.artifactory	Example for Label as property
docker.label.production	Ready for production

Searching for Docker Images

You can search for Docker images by their name, tag or image digest using Artifactory's [Package Search](#) or through the [REST API](#).

Search type: Quick **Package** Archive Property Checksum Remote

Docker V1 V2 Back To Browse

ubu Tag Image Digest

Clear Search

Limit to Specific Repositories

Search Results - 1 Items | [Stash Results](#) ?

Filter by Image AQL Delete Page 1 of 1

Image	Tag	Repository	Modified
ubuntu	latest	docker-dev-local2	Sun Jul 26 13:02:00 UTC 2015

Listing Docker Images

From version 4.4.3, Artifactory supports the following REST API endpoints related to Docker registries:

- [List Docker Images](#) provides a list of Docker images in the specified Artifactory Docker registry. This endpoint mimics the Docker `_catalog` REST API.
- [List Docker Tags](#) provides a list of tags for the specified Docker image.

From version 5.4.6, Artifactory also supports pagination for this endpoint.

Pushing Images to Bintray

Through Artifactory's close integration with JFrog Bintray, you can push Docker images from your Artifactory Docker Registries directly to Bintray. To enable this, make sure your [Bintray credentials](#) are properly configured in your User Profile page.

To push an image to Bintray, use the [Distribution Repository](#).

Deletion and Cleanup

Artifactory natively supports removing tags and repositories and complies with the Docker Hub spec.

Deletion of Docker tags and repositories automatically cleans up any orphan layers that are left (layers not used by any other tag/repository).

Currently, the Docker client does not support DELETE commands, but deletion can be triggered manually. To delete an entire Docker repository using cURL, execute the following command:

```
curl -u<user:password> -X DELETE "<Artifactory URL>/<Docker v2 repository name>/<image namespace>"
```

Or for a specific tag version:

```
curl -u<user:password> -X DELETE "<Artifactory URL>/<Docker v2 repository name>/<image namespace>/<tag>"
```

For example, to remove the latest tag of an Ubuntu repository:

```
//Removing the latest tag from the "jfrog/ubuntu" repository
curl -uadmin:password -X DELETE
"https://artprod.company.com/dockerv2-local/jfrog/ubuntu/latest"
```

Empty Directories

Any empty directories that are left following removal of a repository or tag will automatically be removed during the next folder pruning job (which occurs every 5 minutes by default).

Limiting Unique Tags

To avoid clutter and bloat in your Docker registries caused by many snapshots being uploaded for an image, set the **Max Unique Tags** field in the [Local Docker Repository](#) configuration to limit the number of unique tags.

Migrating from Docker V1 to Docker V2

If you are still using Docker V1, we strongly recommend upgrading to Docker V2. This requires that you migrate any Docker repositories that were created for Docker V1, and is done with a simple cURL endpoint.

For details, please refer to [Migrating a V1 repository to V2](#) under the [Using Docker V1](#) documentation.

Using Docker V1?

This document shows how to use Artifactory with the Docker V2 . If you are using the Docker V1, please refer to [Using Docker V1](#).

Support Matrix

This matrix provides information on features supported as the versions of Artifactory progress.

Artifactory Version	Docker Client Version	Docker V1 API	Docker V2 API	Remote Repositories*	Virtual Repositories*
4.9+	1.12	✓	✓	✓	✓
4.8+	1.11	✓	✓	✓	✓
4.4.3+	1.10	✓	✓	✓	✓
4.1+	1.8+	✓	✓	✓	✓
4.0.2+	1.8	✓	✓	✓	
4.0.0+	1.6+	✓	✓	✓	
4.0.0+	<1.6	✓			

* Supported for Docker V2 API only

Getting Started with Artifactory as a Docker Registry

Overview

There are three main ways you can use Docker with Artifactory and this document describes how to get started with each way.

Please review the brief summary below to decide which is the best way for you to use Docker with Artifactory.

Artifactory SaaS

The easiest way to start using Docker with Artifactory is through an [Artifactory SaaS](#) account.

In this mode, since Artifactory is a hosted service, you do not need to set up a reverse proxy and can create your Docker repositories and start pushing and pulling Docker images.

For more details, please refer to [Getting Started with Artifactory SaaS](#).

Using Docker Compose - 1 Minute Setup

Artifactory can be run in a Docker container preconfigured as a Docker registry.

For more details, please refer to [Using Docker Compose - 1 Minute Setup](#).

Artifactory On-Prem

You can setup your on-prem installation of Artifactory Pro to work with Docker.

Since the Docker client requires a different hostname for each registry you will need to configure a reverse proxy when using this method.

For more details, please refer to [Getting Started with Artifactory Pro On-Prem](#).

Page Contents

- [Overview](#)
- [Getting Started with Artifactory SaaS](#)
 - [Using Docker Client with Artifactory SaaS](#)
 - [Test Your Setup](#)
- [Using Docker Compose - 1 Minute Setup](#)
 - [Complete the Setup](#)

- Test Your Setup
- Getting Started with Artifactory Pro On-Prem
 - The Subdomain Method
 - Configuring Artifactory and Your Reverse Proxy
 - Test Your Setup
 - The Ports Method
 - Configuring Artifactory and Your Reverse Proxy
 - Configuring Your Docker Client
 - Test Your Setup

Getting Started with Artifactory SaaS

Using Docker repositories with [Artifactory SaaS](#) is quick and easy to use.

Since, with Artifactory SaaS, you are using Artifactory as a hosted service, there is no need to configure Artifactory with a reverse proxy.

The example at the end of this section shows a complete process of creating a Docker repository, logging in, pulling an image and pushing an image.

Using Docker Client with Artifactory SaaS

To use the Docker client with one of your Artifactory SaaS Docker repositories, you can use the native Docker client to login to each Docker repository, pull, and push images as shown in the following example:

- Login to your repository use the following command with your Artifactory SaaS credentials

```
docker login ${server-name}-{repo-name}.jfrog.io
```

- Pull an image using the following command

```
docker pull ${server-name}-{repo-name}.jfrog.io/<image name>
```

- To push an image, first tag it and then use the push command

```
docker tag <image name> ${server-name}-{repo-name}.jfrog.io/<image name>
docker push ${server-name}-{repo-name}.jfrog.io/<image name>
```

Test Your Setup

You can test your setup with this example that assumes you are using an Artifactory SaaS server named "**acme**".

The scenario it demonstrates is:

- Pulling the "hello-world" Docker image
- Logging into your local Docker repository
- Retagging the "hello-world" image, and the pushing it into your local Docker repository

Start by creating a [local Docker repository](#) called `dockerv2-local`.

- Pull the "hello-world" image

```
docker pull hello-world
```

- Login to repository `dockerv2-local`


```
docker login acme-dockerv2-local.jfrog.io
```

- Tag the "hello-world" image

```
docker tag hello-world acme-dockerv2-local.jfrog.io/hello-world
```

- Push the tagged "hello-world" image to dockerv2-local

```
docker push acme-dockerv2-local.jfrog.io/hello-world
```

Using Docker Compose - 1 Minute Setup

Artifactory may easily be installed as a Docker registry running in Docker. This is the easiest way to use Artifactory as a Docker registry on-premises. The installation spins up the following three containers:

- Artifactory Pro
- NGINX proxy that uses a self-signed certificate and is configured for access using the sub-domain method
- A Postgres database

To spin up this installation run the following command:

```
curl -L  
'https://bintray.com/api/v1/content/jfrog/run/art-compose/$latest/art-comp  
ose?bt_package=art-compose' | sudo bash
```

Complete the Setup

To complete the setup, invoke the **onboarding wizard** by running Artifactory in your browser at `http://<HOST_NAME>/artifactory`.

- Activate Artifactory with your license key. If you do not have a license you can get a free 30 day [Trial license](#).
- You may set the Admin password or skip to accept the default
- If necessary, configure your network proxy or just skip this step (you may [configure a proxy server](#) at any time later)
- At **Create Repositories**, select **Docker** and continue to complete the wizard

Sub-domains method

We use this method so you will not need to change the reverse proxy configuration for each new Docker repository created.

Finally, follow the steps below:

1. You need to add the following to your DNS or `/etc/hosts` file:

```
<ip-address> docker-local.artifactory docker-remote.artifactory  
docker.artifactory artifactory
```

2. Since the certificate is self-signed, you need to import it to your Docker certificate trust store as described in the [Docker documentation](#). Alternatively, you can configure the Docker client to work with an insecure registry by adding the following line to your `/etc/default/docker` file (you may need to create the file if it does not already exist):

```
DOCKER_OPTS="$DOCKER_OPTS --insecure-registry docker-local.artifactory
--insecure-registry docker-remote.artifactory --insecure-registry
docker.artifactory"
```

- Restart your Docker daemon/engine to apply the insecure registry flag (if self-signed certificate is imported, you do not need to restart the Docker daemon/engine).

Test Your Setup

You can test your setup with this example .

The scenario it demonstrates is:

- Pulling the "hello-world" Docker image
- Logging into your virtual Docker repository
- Retagging the "hello-world" image, and the pushing it into your virtual Docker repository

The Artifactory Docker registry is already configured with a virtual repository called `docker.artifactory`.

- Pull the "hello-world" image

```
docker pull hello-world
```

- Login to repository "docker.artifactory"

```
docker login docker.artifactory
```

- Tag the "hello-world" image

```
docker tag hello-world docker.artifactory/hello-world
```

- Push the tagged "hello-world" image to docker.artifactory

```
docker push docker.artifactory/hello-world
```

Getting Started with Artifactory Pro On-Prem

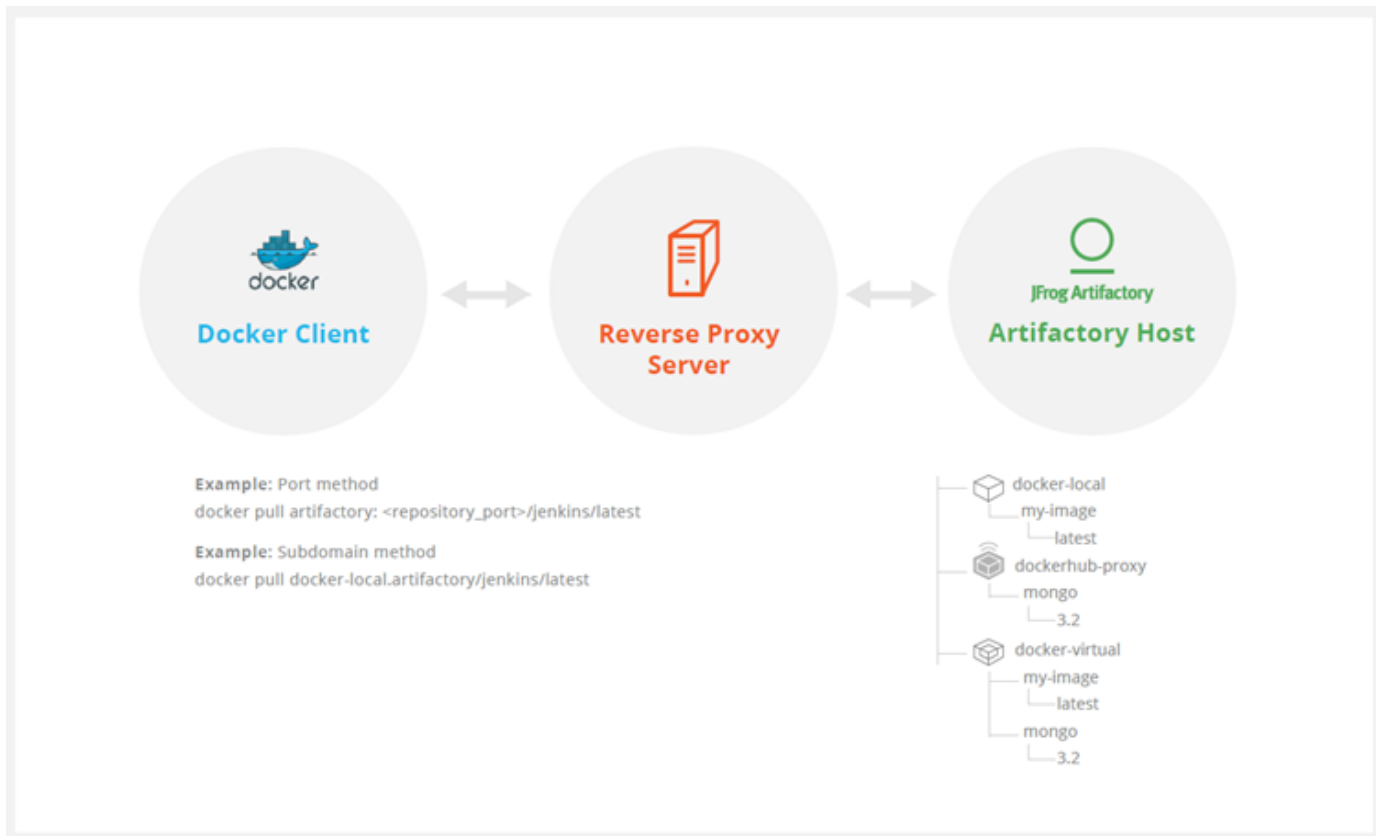
Using Artifactory Pro On-Prem with Docker requires a reverse proxy due to the following limitations of the Docker client:

1. You cannot use a context path when providing the registry path (e.g `localhost:8081/artifactory` is not valid)
2. Docker will only send basic HTTP authentication when working against an HTTPS host

Therefore, you need a reverse proxy to map Docker commands to Docker registries in Artifactory using either the **subdomain method** or the **ports method**.

Testing or evaluating?

If you are currently only testing or evaluating using Artifactory with Docker, we recommend [running Artifactory as a Docker container](#) which is easily installed and comes with a proxy server and Docker registries pre-configured out-of-the-box. You can be up and running in minutes.



With the ports method, a port number mapped to each Artifactory Docker registry. While this is an easy way to get started, you will need to modify your reverse proxy configuration and add a new mapping for each new Docker registry you define in Artifactory. In addition, firewalls and other restrictions by your IT department may restrict port numbers making the ports method not feasible.

With the subdomain method, you only need to configure your reverse proxy once, and from then on, the mapping from Docker commands to Docker registries in Artifactory is dynamic and requires no further modification of your reverse proxy configuration.

We recommend to use the subdomain method since it will require one time effort.

The Subdomain Method

Getting started with Docker and your on-prem Artifactory Pro installation using the subdomain method involves two basic steps:

1. Configuring Artifactory and your reverse proxy.
2. Configuring your Docker client.

Configuring Artifactory and Your Reverse Proxy

To configure Artifactory and your reverse proxy using the subdomain method, carry out the following steps:

1. Make sure Artifactory is **up and running**, and is **activated with a valid license**.
2. Create your **local Docker repository**. In our example below we will use a repository named **docker-local**.
3. Make sure you have a reverse proxy server up and running.
4. Obtain a **wildcard** SSL certificate or use a wildcard self-signed certificate.

Make sure your certificate matches the **Artifactory hostname** used in your reverse proxy configuration. In our example below we will use **art.local**.

5. Configure your reverse proxy. Artifactory's [Reverse Proxy Configuration Generator](#) can generate your complete reverse proxy configuration file for supported servers. All you need to do is fill in the fields in according to how your reverse proxy is set up while making sure to:
 - a. Use the correct **Artifactory hostname** in the **Public Server Name** field (in our example this will be **art.local**)
 - b. Select **Subdomain** as the **Reverse Proxy Method** under **Docker Reverse Proxy Settings**

NGINX

Copy the code snippet generated by the [configuration generator](#) into your `artifactory-nginx.conf` file, and place it in your `/etc/n`

nginx/sites-available directory.

Create the following symbolic link.

```
sudo ln -s /etc/nginx/sites-available/artifactory-nginx.conf
/etc/nginx/sites-enabled/artifactory-nginx.conf
```

Apache HTTPD

Copy the code snippet generated by the [configuration generator](#) into your *artifactory-apache.conf* file and place it in you */etc/apache2/sites-available* directory.

Create the following symbolic link:

```
sudo ln -s /etc/apache2/sites-available/artifactory-apache.conf
/etc/apache2/sites-enabled/artifactory-apache.conf
```

Configuring Your Docker Client

To configure your Docker client, carry out the following steps

1. Add the following to your DNS or to the client's */etc/hosts* file:

```
<ip-address> docker-local.art.local
```

2. Since the certificate is self-signed, you need to import it to your Docker certificate trust store as described in the [Docker documentation](#). Alternatively, you can configure the Docker client to work with an insecure registry by adding the following line to your */etc/default/docker* file (you may need to create the file if it does not already exist):

```
DOCKER_OPTS="$DOCKER_OPTS --insecure-registry docker-local.art.local"
```

3. Restart your Docker daemon/engine to apply the insecure registry flag (if self-signed certificate is imported, you do not need to restart the Docker daemon/engine).

Test Your Setup

To verify your reverse proxy is configured correctly, run the following command making sure that the return code is 200:

```
curl -I -k -v https://<artifactory url>
```

Run the following commands to ensure your proxy configuration is functional and can communicate with Artifactory:

- Pull the "hello-world" image

```
docker pull hello-world
```

- Login to repository docker-local

```
docker login docker-local.art.local
```

- Tag the "hello-world" image

```
docker tag hello-world docker-local.art.local/hello-world
```

- Push the tagged "hello-world" image to docker-local

```
docker push docker-local.art.local/hello-world
```

The Ports Method

Getting started with Docker and your on-prem Artifactory Pro installation using the ports method involves two basic steps:

1. [Configuring Artifactory and your reverse proxy.](#)
2. [Configuring your Docker client.](#)

Configuring Artifactory and Your Reverse Proxy

To configure Artifactory and your reverse proxy using the ports method, carry out the following steps:

1. Make sure Artifactory is [up and running](#), and is [activated with a valid license](#).
2. Create your [local Docker repository](#). In our example below we will use a repository named **docker-local**.
3. Make sure you have a reverse proxy server up and running.
4. Obtain an SSL certificate or use a Self-Signed certificate that can be generated following this [example](#).

Make sure your certificate matches the **Artifactory hostname** used in your reverse proxy configuration. In our example below we will use **art.local**.

5. Configure your reverse proxy. Artifactory's [Reverse Proxy Configuration Generator](#) can generate your complete reverse proxy configuration file for supported servers. All you need to do is fill in the fields in according to how your reverse proxy is set up while making sure to:
 - a. Use the correct **Artifactory hostname** in the **Public Server Name** field
 - b. Select **Ports** as the **Reverse Proxy Method** under **Docker Reverse Proxy Settings**. In the example below, we will use port **5001** to bind repository **docker-local**.

NGINX

For Artifactory to work with Docker, the preferred web server is **NGINX v1.3.9** and above.

First, you need to create a self-signed certificate for NGINX [as described here for Ubuntu](#).

Then use Artifactory's [Reverse Proxy Configuration Generator](#) to generate the configuration code snippet for you.

Copy the code snippet into your `artifactory-nginx.conf` file and place it in your `/etc/nginx/sites-available` directory.

Finally, create the following symbolic link:

```
sudo ln -s /etc/nginx/sites-available/artifactory-nginx.conf
/etc/nginx/sites-enabled/artifactory-nginx.conf
```

Apache HTTPD

Install [Apache HTTP server as a reverse proxy](#) and then install the [required modules](#).

Create the following symbolic link:

```
sudo ln -s /etc/apache2/mods-available/slotmem_shm.load
/etc/apache2/mods-enabled/slotmem_shm.load
```

Similarly, create corresponding symbolic links for:

- headers
- proxy_balancer
- proxy_load
- proxy_http

- proxy_connect
- proxy_html
- rewrite.load
- ssl.load
- lbmethod_byrequests.load

Then use Artifactory's [Reverse Proxy Configuration Generator](#) to generate the configuration code snippet for you. Copy the code snippet into your `artifactory.conf` file and place it in your `/etc/apache2/sites-available` directory. HAProxy

First, you need to create a self-signed certificate for HAProxy [as described here for Ubuntu](#).

Then, copy the code snippet below into your `/etc/haproxy/haproxy.cfg` file. After editing the file as described in the snippet, you can test your configuration using the following command:

```
haproxy -f /etc/haproxy/haproxy.cfg -c
```

HAProxy v1.5 Configuration

```
# haproxy server configuration
# version 1.0
# History
#
-----
# Features enabled by this configuration
# HA configuration
# port 80, 443 Artifactory GUI/API
#
# This uses ports to distinguish artifactory docker repositories
# port 443 docker-virtual (v2) docker v1 is redirected to
docker-dev-local.
# port 5001 docker-prod-local (v1); docker-prod-local2 (v2)
# port 5002 docker-dev-local (v1); docker-dev-local2 (v2)
#
# Edit this file with required information enclosed in <...>
# 1. certificate and key
# 2. artifactory-host
# 3 replace the port numbers if needed
#
-----
global
    log 127.0.0.1 local0
    chroot /var/lib/haproxy
    maxconn 4096
    user haproxy
    group haproxy
    daemon
    tune.ssl.default-dh-param 2048
    stats socket /run/haproxy/admin.sock mode 660 level admin
defaults
    log global
    mode http
    option httplog
    option dontlognull
```

```

option redispatch
option forwardfor
option http-server-close
maxconn 4000
timeout connect 5000
timeout client 50000
timeout server 50000
errorfile 400 /etc/haproxy/errors/400.http
errorfile 403 /etc/haproxy/errors/403.http
errorfile 408 /etc/haproxy/errors/408.http
errorfile 500 /etc/haproxy/errors/500.http
errorfile 502 /etc/haproxy/errors/502.http
errorfile 503 /etc/haproxy/errors/503.http
errorfile 504 /etc/haproxy/errors/504.http
frontend normal
    bind *:80
    bind *:443 ssl crt </etc/ssl/certs/server.bundle.pem>
    mode http
    option forwardfor
    requirep ^([\^ \:]*)\ /v2(.*) \1\
/artifactory/api/docker/docker-virtual/v2\2
    reqadd X-Forwarded-Proto:\ https if { ssl_fc }
    option forwardfor header X-Real-IP
    default_backend normal

# if only need to access the docker-dev-local2 then skip this section.
Docker-virtual can be configured to deploy to docker-dev-local2
frontend dockerhub
    bind *:5000 ssl crt </etc/ssl/certs/server.bundle.pem>
    mode http
    option forwardfor
    option forwardfor header X-Real-IP
    requirep ^([\^ \:]*)\ /v2(.*) \1\
/artifactory/api/docker/docker-remote/v2\2
    reqadd X-Forwarded-Proto:\ https if { ssl_fc }
    default_backend normal

# if only need to access the docker-dev-local2 then skip this section.
Docker-virtual can be configured to deploy to docker-dev-local2
frontend dockerprod
    bind *:5001 ssl crt </etc/ssl/certs/server.bundle.pem>
    mode http
    option forwardfor
    option forwardfor header X-Real-IP
    requirep ^([\^ \:]*)\ /v1(.*) \1\
/artifactory/api/docker/docker-prod-local/v1\2
    requirep ^([\^ \:]*)\ /v2(.*) \1\
/artifactory/api/docker/docker-prod-local2/v2\2
    reqadd X-Forwarded-Proto:\ https if { ssl_fc }
    default_backend normal

# if only need to access the docker-dev-local2 then skip this section.
Docker-virtual can be configured to deploy to docker-dev-local2

```

```
frontend dockerdev
    bind *:5002 ssl crt </etc/ssl/certs/server.bundle.pem>
    mode http
    option forwardfor
    option forwardfor header X-Real-IP
    requirep ^([\^ \ :]*)\ /v1(.*$) \1\
/artifactory/api/docker/docker-dev-local/v1\2
    requirep ^([\^ \ :]*)\ /v2(.*$) \1\
/artifactory/api/docker/docker-dev-local2/v2\2
    reqadd X-Forwarded-Proto:\ https if { ssl_fc }
    default_backend normal

# Artifactory Non HA Configuration
# i.e server artifactory 198.168.1.206:8081
#
backend normal
    mode http
    server <artifactory-host> <artifactory-host ip
address>:<artifactory-host port>

#
# Artifactory HA Configuration
# Using default failover interval - rise = 2; fall =3 3; interval - 2
seconds
# backend normal
#     mode http
#     balance roundrobin
#     option httpchk OPTIONS /
#     option forwardfor
#     option http-server-close
#     appsession JSESSIONID len 52 timeout 3h
#     server <artifactory-host-ha1> <artifactory-host ip
```



```
address>:<artifactory-host port>
#   server <artifactory-host-ha2> <artifactory-host ip
address>:<artifactory-host port>
```

Configuring Your Docker Client

To configure your Docker client, carry out the following steps

1. Add the following to your DNS or to the client's `/etc/hosts` file:

```
<ip-address> art.local
```

2. Since the certificate is self-signed, you need to import it to your Docker certificate trust store as described in the [Docker documentation](#). Alternatively, you can configure the Docker client to work with an insecure registry by adding the following line to your `/etc/default/docker` file (you may need to create the file if it does not already exist):

```
DOCKER_OPTS="$DOCKER_OPTS --insecure-registry art.local:5001"
```

3. Restart your Docker engine.

Test Your Setup

To verify your reverse proxy is configured correctly, run the following command:

```
// Make sure the following results in return code 200
curl -I -k -v https://<artifactory url>
```

Run the following commands to ensure your proxy configuration is functional and can communicate with Artifactory. In this example, we will pull down a Docker image, tag it and then deploy it to our `docker-local` repository that is bound to **port 5001**:

```
// Pull the "hello-world" image
docker pull hello-world

// Login to repository docker-local
docker login art-local:5001

// Tag the "hello-world" image
docker tag hello-world art-local:5001/hello-world

// Push the tagged "hello-world" image to docker-local
docker push art-local:5001/hello-world
```

Testing With a Self-signed Certificate

1. Since the certificate is self-signed, you need to import it to your Docker certificate trust store as described in the [Docker documentation](#). Alternatively, you can configure the Docker client to work with an insecure registry by adding the following line to your `/etc/default/docker` file (you may need to create the file if it does not already exist).

```
DOCKER_OPTS="$DOCKER_OPTS --insecure-registry docker-local.art.local"
```

- Restart your Docker daemon/engine to apply the insecure registry flag (if self-signed certificate is imported, you do not need to restart the Docker daemon/engine).
- Use the steps above to interact with the Artifactory Docker Registry

Advanced Topics

Overview

This page provides some advanced topics for using Docker with Artifactory.

Page Contents

- [Overview](#)
- [Using a Self-signed SSL Certificate](#)
- [Using Your Own Certificate](#)
- [Setting Your Credentials Manually](#)
- [Authenticating via OAuth](#)

Using a Self-signed SSL Certificate

You can use self-signed SSL certificates with `docker push/pull` commands, however for this to work, you need to specify the `--insecure-registry` daemon flag for each insecure registry.

For full details please refer to the [Docker documentation](#).

For example, if you are running Docker as a service, edit the `/etc/default/docker` file, and append the `--insecure-registry` flag with your registry URL to the `DOCKER_OPTS` variable as in the following example:

Edit the DOCKER_OPTS variable

```
DOCKER_OPTS="-H unix:///var/run/docker.sock --insecure-registry  
artprod.company.com"
```

For this to take effect, you need to restart the Docker service.

If you are using **Boot2Docker**, please refer to the **Boot2Docker** documentation for [Insecure Registry](#).

If you do not make the required modifications to the `--insecure-registry` daemon flag, you should get the following error:

Error message

```
v2 ping attempt failed with error: Get https://artprod.company.com/v2/:  
x509: cannot validate certificate for artprod.company.com because it  
doesn't contain any IP SANs
```

Using Your Own Certificate

The NGINX configuration provided with Artifactory out-of-the-box references the internally bundled certificate and key which you may replace with your own certificate and key.

For details, please refer to [Using Your Own Certificate](#).

Setting Your Credentials Manually

If you are unable to log in to Docker, you may need to set your credentials manually.

Manually setting your Docker credentials

The Docker command line tool supports authenticating sensitive operations, such as push, with the server using basic HTTP authentication.

To enforce authenticated access to docker repositories you need to provide the following parameters to the Docker configuration file.

- The Docker endpoint URL (must use HTTPS for basic authentication to work)

- Your Artifactory username and password (formatted `username:password`) as [Base64](#) encoded strings
- Your email address

You can use the following command to get these strings directly from Artifactory and copy/paste them into your `~/.dockercfg` file:

sudo

If you are using Docker commands with "sudo" or as a root user (for example after installing the Docker client), note that the Docker configuration file should be placed under `/root/.dockercfg`

Getting `.dockercfg` entries directly from Artifactory

```
$ curl -uadmin:password "https://artprod.company.com/<v1|v2>/auth"
{
  "https://artprod.company.com" : {
    "auth" : "YWRtaW46QVA1N050aHZTMnM5Qk02Rkr5RjNBVmF4TVF1",
    "email" : "admin@email.com"
  }
}
```

The Docker configuration file may contain a separate authentication block for each registry that you wish to access.

Below is an example with two URL endpoints:

```
{
  "https://artprod.company.com": {
    "auth": "YWRtaW46cGFzc3dvcmQ=",
    "email": "myemail@email.com"
  },
  "https://artprod2.company.com": {
    "auth": "YWRtaW46cGFzc3dvcmQ=",
    "email": "myemail@email.com"
  }
}
```

Authenticating via OAuth

From version 4.4, Artifactory supports authentication of the Docker client using OAuth through the default GitHub OAuth provider. When authenticating using OAuth you will not need to provide additional credentials to execute `docker login` with Artifactory.

To set up OAuth authentication for your Docker client, execute the following steps:

- Under **General OAuth Settings**, make sure **Auto Create Artifactory Users** is checked to make sure a user record is created for you first time you log in to Artifactory with OAuth.
- Log in to Artifactory with OAuth using your Git Enterprise account

Once you are logged in to Artifactory through your Git Enterprise OAuth account, your Docker client will automatically detect this and use OAuth for authentication, so you do not need to provide additional credentials.

Working with Docker Content Trust

Overview

Notary is Docker's platform to provide trusted delivery of content by signing images that are published. A content publisher can then provide the corresponding signing keys that allow users to verify that content when it is consumed. Artifactory fully supports working with Docker Notary to ensure that Docker images uploaded to Artifactory can be signed, and then verified when downloaded for consumption. When the Docker client is configured to work with Docker Notary, after pushing an image to Artifactory, the client notifies the Notary to sign the image before assigning it a tag.

Artifactory supports hosting signed images without the need for any additional configuration.

Page Contents

- Overview
- Configuring Docker Notary and Docker Client
 - Configuring Your Hosts File
 - Configuring the Notary Server
 - Configuring the Docker Client
- Test Your Setup

Configuring Docker Notary and Docker Client

There is no configuration needed in Artifactory in order to work with trusted Docker images. However, in the setup instructions below, we do recommend testing your configuration by signing Artifactory and running it in a container.

To configure the Docker Notary and client to work with Artifactory, execute the following main steps:

- Configure your hosts file
- Configure the Notary server and run it as a container
- Configure the Docker client

Configuring Your Hosts File

If you are not working with a DNS, add the following entries to your `/etc/hosts` file:

```
sudo sh -c 'echo "<Host IP> <Notary Server Name>" >> /etc/hosts'
sudo sh -c 'echo "<Host IP> <Artifactory Server Name>" >> /etc/hosts'
```

Configuring the Notary Server

Create a directory for your Notary server. In the code snippets below we will use `notarybox`.

Create a dockerfile with the following content:

```
FROM debian:jessie

ADD https://get.docker.com/builds/Linux/x86_64/docker-1.9.1 /usr/bin/docker
RUN chmod +x /usr/bin/docker \
    && apt-get update \
    && apt-get install -y \
    tree \
    vim \
    git \
    ca-certificates \
    --no-install-recommends

WORKDIR /root
RUN git clone https://github.com/docker/notary.git && \
    cp /root/notary/fixtures/root-ca.crt \
    /usr/local/share/ca-certificates/root-ca.crt && \
    update-ca-certificates

ENTRYPOINT ["bash"]
```

Use a private certificate

This configuration runs with a public certificate. Any Docker client running with the same public certificate may be able to access your

Notary server.

For a secure setup, we recommend replacing it with your organization's private certificate by replacing the public `root-ca.crt` certificate file with your private certificate under `/root/notary/fixtures` on your Notary server, and under `/usr/local/share/certificates` on the machine running your Docker client.

Build the test image:

```
docker build -t [image name] [path to dockerfile]
```

If you are running the build in your dockerfile directory, you can just use "." as the path to the dockerfile

Start the Notary server:

To start the Notary server, you first need to have [Docker Compose](#) installed.

Then execute the following steps:

```
cd notarybox
git clone -b trust-sandbox https://github.com/docker/notary.git
cd notary
docker-compose build
docker-compose up -d
```

Configuring the Docker Client

To connect the Notary server to the Docker client you need to enable the Docker content trust flag and add the Notary server URL as follows:

```
export DOCKER_CONTENT_TRUST=1
export DOCKER_CONTENT_TRUST_SERVER=https://notaryserver:4443
```

Test Your Setup

The example below demonstrates setting up the Notary server and Docker client, signing an image and the pushing it to Artifactory, with the following assumptions:

- Notary server and Artifactory run on localhost (127.0.0.1)
- Notary server is in directory `notarybox`
- Working without a DNS (so we need to configure the `hosts` file)
- Notary server name is `notaryserver`
- Artifactory server name is `artifactory-registry`
- Docker Compose is installed

Set up the IP mappings

```
sudo sh -c 'echo "127.0.0.1 notaryserver" >> /etc/hosts'
sudo sh -c 'echo "127.0.0.1 artifactory-registry" >> /etc/hosts'
```

Create the Dockerfile

```
FROM debian:jessie

ADD https://get.docker.com/builds/Linux/x86_64/docker-1.9.1 /usr/bin/docker
RUN chmod +x /usr/bin/docker \
    && apt-get update \
    && apt-get install -y \
    tree \
    vim \
    git \
    ca-certificates \
    --no-install-recommends

WORKDIR /root
RUN git clone -b trust-sandbox https://github.com/docker/notary.git && \
    cp /root/notary/fixtures/root-ca.crt \
    /usr/local/share/ca-certificates/root-ca.crt && \
    update-ca-certificates

ENTRYPOINT ["bash"]
```

Note that this example uses the public `root-ca.crt` certificate.

Navigate to the Dockerfile location and build the test image

```
docker build -t notarybox .
```

Run Artifactory as a container

```
docker pull
jfrog-docker-reg2.bintray.io/jfrog/artifactory-registry:<version>
docker run -d --name artifactory-registry -p 80:80 -p 8081:8081 -p 443:443
-p 5000-5002:5000-5002
jfrog-docker-reg2.bintray.io/jfrog/artifactory-registry:<version>
```

Access your Artifactory instance (at <http://localhost:8081/artifactory>) and [activate it](#) with an Artifactory Pro license.

For more details on running Artifactory as a Docker container, please refer to [TEMP - Installing with Docker](#).

Start your container

In this step you will start the container with the `dockerfile` you created earlier, and link it to your Notary server and Artifactory.

```
docker run -it -v /var/run/docker.sock:/var/run/docker.sock --link
notary_server_1:notaryserver --link
artifactory-registry:artifactory-registry notarybox
```

Pull an image for testing

```
docker pull docker/trusttest
```

After you have pulled the image, you need to `docker login artifactory-registry:5002/v2`

Configure the Docker client

```
export DOCKER_CONTENT_TRUST=1
export DOCKER_CONTENT_TRUST_SERVER=https://notaryserver:4443
```

Tag the image you pulled for testing and push it to Artifactory

```
docker tag docker/trusttest artifactory-registry:5002/test/trusttest:latest
docker push artifactory-registry:5002/test/trusttest:latest
```

You will be asked to enter the root key passphrase. This will be needed every time you push a new image while the `DOCKER_CONTENT_TRUST` flag is set.

The root key is generated at: `/root/.docker/trust/private/root_keys`

You will also be asked to enter a new passphrase for the image. This is generated at `/root/.docker/trust/private/tuf_keys/[registry name] /[imagepath]`

Using Docker V1

Overview

This page describes how to use Artifactory with the Docker V1 Registry API. If you are using the Docker V2 Registry API, please refer to [Docker Registry](#).

For general information on using Artifactory with Docker, please refer to [Artifactory as a Docker Registry](#).

Getting Started with Artifactory and Docker

Artifactory supports Docker transparently, meaning you can point the Docker client at Artifactory and issue push, pull and other commands in exactly the same way that you are used to when working directly with a private registry or Docker Hub.

To get started using Docker with Artifactory you need to execute the following steps:

1. [Set up a web server as a reverse proxy](#)
2. [Create a local repository](#)
3. [Set up authentication](#)
4. [Push and pull images](#)

The [screencast](#) at the end of this section provides a demonstration.

1. Setting up NGINX as a Reverse Proxy

Artifactory can only be used with Docker through a reverse proxy due to the following limitations of the Docker client:

1. You cannot provide a context path when providing the registry path (e.g `localhost:8081/artifactory` is not valid)
2. Docker will only send basic HTTP authentication when working against an HTTPS host

For Artifactory to work with Docker, the preferred web server is **NGINX v1.3.9** and above configured as a reverse proxy.

For other supported web servers, please refer to [Alternative Proxy Servers](#).

Below is a sample configuration for NGINX which configures SSL on port 443 to a specific local repository in Artifactory (named `docker-local`) on a server called `artprod.company.com`.

Using Docker v1, Docker client v1.10 and Artifactory 4.4.3 known issue.

To avoid incompatibility when using Docker V1 with Docker 1.10, use the NGINX configuration displayed below and not the NGINX configuration generated by Artifactory v4.4.3.

▼ [NGINX Configuration for Docker V1](#)

This code requires NGINX to support chunked transfer encoding which is available from NGINX v1.3.9.


```

[...]

http {

    ##
    # Basic Settings
    ##
    [...]

    server {
        listen 443;
        server_name artprod.company.com;

        ssl on;
        ssl_certificate
/etc/ssl/certs/artprod.company.com.crt;
        ssl_certificate_key
/etc/ssl/private/artprod.company.com.key;

        access_log
/var/log/nginx/artprod.company.com.access.log;
        error_log
/var/log/nginx/artprod.company.com.error.log;

        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Original-URI
$request_uri;
        proxy_read_timeout 900;

        client_max_body_size 0; # disable any
limits to avoid HTTP 413 for large image uploads

        # required to avoid HTTP 411: see Issue
#1486
(https://github.com/docker/docker/issues/1486)
        chunked_transfer_encoding on;

        location /v1 {
            proxy_pass
http://artprod.company.com:8081/artifactory/api/do
cker/docker-local/v1;
        }
    }
}

```

If you want to use multiple Docker repositories, you need to copy this configuration and bind different ports to each local repository in Artifactory. For details, please refer to [Port Bindings](#).

Repository URL prefix

When accessing a Docker repository through Artifactory, the repository URL must be prefixed with **api/docker** in the path. For details, please refer to [Docker Repository Path and Domain](#).

2. Creating a Local Docker Repository

This is done in the same way as when [configuring a local repository](#) to work with Docker V2, however, in the Docker Settings section, you should make sure to select V1 as the Docker API version.

Docker Settings

API Version

V1

V2

Force Authentication [?](#)

Page Contents

- Overview
- Getting Started with Artifactory and Docker
 - 1. Setting up NGINX as a Reverse Proxy
 - 2. Creating a Local Docker Repository
 - Working with Artifactory SaaS
 - 3. Setting Up Authentication
 - 4. Pushing and Pulling Images
 - Watch the Screencast
- Browsing Docker Repositories
 - Viewing the Docker Images Tree
 - Viewing Individual Docker image Information
 - Searching for Docker Images
 - Promoting Docker Images with V1
- Migrating a V1 repository to V2
- Deletion and Cleanup
- Advanced Topics
 - Using a Self-signed SSL Certificate
 - Alternative Proxy Servers
 - Apache Configuration
 - Port Bindings
 - Docker Repository Path and Domain
- Support Matrix

Working with Artifactory SaaS

▼ [Click here to expand...](#)

Due to limitations of the Docker client, in Artifactory SaaS there is a special configuration for each server with a sub-domain.

You need to create a new Docker enabled local repository named **docker-local**.

Then, use the following address when working with the Docker client: "**`\${account_name}.jfrog.io**"

3. Setting Up Authentication

When using Artifactory with Docker V1, you need to set your credentials manually by adding the following section to your `~/.docker/config.json` file.

~/docker/config.json

```
{
  "auths" :{
    "https://artprod.company.com" : {
      "auth": "<USERNAME>:<PASSWORD> (converted to base 64)",
      "email": "youremail@email.com"
    },
    "https://artdev.company.com" : {
      "auth": "<USERNAME>:<PASSWORD> (converted to base 64)",
      "email": "youremail@email.com"
    }
  }
}
```

4. Pushing and Pulling Images

Pushing and pulling images when using Docker V1 is done in the same way as when using Docker V2. Please refer to [Pushing and Pulling Images](#) under the Docker Repositories page.

Watch the Screencast

Once you have completed the above setup you should be able to use the Docker client to transparently push images to and pull them from Docker repositories in Artifactory. You can see this in action in the screencast below.

Browsing Docker Repositories

Artifactory stores docker images in a layout that is made up of 2 main directories:

- **.images:** Stores all the flat docker images.
- **repositories:** Stores all the repository information with tags (similar to how repositories are stored in the Docker Hub).

In addition, Artifactory annotates each deployed docker image with two properties:

- **docker.imageld:** The image id
- **docker.size:** The size of the image in bits

Deployed tags are also annotated with two properties:

- **docker.tag.name:** The tag name
- **docker.tag.content:** The id of the image that this tag points to

The screenshot shows the Artifactory Repository Browser interface. On the left, a tree view displays the repository structure, including folders like 'docker-1', 'docker-2', and 'docker-3', and sub-folders like '.images' and 'repositories'. The 'repositories' folder is expanded, showing a list of repository IDs, with '9fd3c8c9af32' selected. On the right, the 'Properties' tab is active for the selected repository. It shows a table of properties with two columns: 'Property' and 'Value(s)'. The properties listed are 'docker.size' with a value of '1895' and 'docker.imageld' with a long alphanumeric string. There are also input fields for adding new properties and a 'Filter by Property' search box.

Property	Value(s)
docker.size	1895
docker.imageld	9fd3c8c9af32dddb1793ccb5f6535e12d735eacae16f8f8c4214f42f3fe3d29

Artifact Repository Browser Set Me Up Deploy

Tree Simple Compress Empty Folders

- ▶ bower-local
- ▶ debian-local
- ▶ docker-local
- ▶ dockerv1
 - ▶ .images
 - ▶ repositories
 - ▶ library
 - ▶ ubuntu
 - ▶ latest
 - tag.json

- ▶ ext-release-local
- ▶ ext-snapshot-local
- ▶ gem1
- ▶ generic-local

tag.json View Download Actions

General Effective Permissions **Properties** Watchers Builds Governance

Add: Property Property Set

Name* Value

Recursive

Filter by Property Delete < page 1 of 1 >

Property	Value(s)
docker.tag.name	latest
docker.tag.content	6d4946999d4fb403f40e151ecbd13cb866da125431eb1df0cfd4dc72674e3c6

Viewing the Docker Images Tree

Artifactory lets you view the complete images tree for a specific image directly from the UI in a similar way to what you would get from the `docker images --tree` command.

In the **Artifacts** module **Tree Browser**, drill down to select the image you want to inspect. The metadata is displayed in the **Docker Ancestry** tab.

 6d4946999d4f

General Docker Info **Docker Ancestry** Effective Permissions Properties Watchers

- |_428b411c28f0 Virtual Size: 188.1 Mbit
- |_435050075b3f Virtual Size: 188.3 Mbit
- |_9fd3c8c9af32 Virtual Size: 188.3 Mbit
- |_6d4946999d4f Virtual Size: 188.3 Mbit

Viewing Individual Docker image Information

In the **Artifacts** module **Tree Browser**, drill down to select image you want to inspect. The metadata is displayed in the **Docker Info** tab.

General

Docker Info

Docker Ancestry

Effective Permissions

Properties

Watchers

Package Info

Image Id:	6d4946999d4fb403f40e151ecbd13cb866da125431eb1df0cdfd4dc72674e3c6
Parent Id:	9fd3c8c9af32dddb1793ccb5f6535e12d735eacae16f8f8c4214f42f33fe3d29
Created:	2015-06-12T15:32:30.680894574Z
Container:	9201d6220b01338011f2f21c28429d2155625625392e6654fe378c2772cc46bd
Docker Version:	1.6.0
Architecture:	amd64
OS:	linux
Size:	0 bits (0 bit)

Config

Hostname:	d3659c5e113e
DomainName:	
User:	
Memory:	0
MemorySwap:	0
CpuShares:	0
AttachStdin:	false
AttachStdout:	false
AttachStderr:	false
Tty:	false
OpenStdin:	false
StdinOnce:	false

Searching for Docker Images

In addition to other properties related to Docker repositories, you can also search for repositories using a property called `docker.repoName`, which represents the repository name (e.g., "library/ubuntu").

_index_images.json

View Download Actions

General

Effective Permissions

Properties

Watchers

Builds

Governance

Add: Property Property Set

Name *

Value

Add

 Recursive

Filter by Property

Delete < page 1 of 1 >

Property	Value(s)
docker.repoName	library/ubuntu

Promoting Docker Images with V1

Promoting Docker images with Docker V1 is done in exactly the same way as when [Promoting Images with Docker V2](#).

Migrating a V1 repository to V2

We recommend using Docker V2 repositories when possible (provided your Docker client is version 1.6 and above).

If you have an existing Docker V1 repository, you can migrate its content into a V2 repository using the following endpoint with cURL:

```
POST api/docker/<repoKey>/v1/migrate
{
  "targetRepo" : "<targetRepo>",
  "dockerRepository" : "<dockerRepository>",
  "tag" : "<tag>"
}
```

where:

<repoKey>	Source repository key (For example, <i>docker-local</i> as used in this page)
<targetRepo>	The target Docker V2 repository to migrate to (For example, <i>docker-local2</i> as used in this page). The repository should be created before running the <i>migrate</i> endpoint.
<dockerRepository>	An optional docker repository name to migrate, if null - the entire source repository will be migrated. Default: ""
<tag>	An optional tag name to promote, if null - the entire docker repository will be promoted. Default: ""

An example for migrating the docker image "*jfrog/ubuntu*" with all of it's tags from *docker-local* to *docker-local2* using cURL would be:

```
curl -i -uadmin:password -X POST
"http://localhost:8081/artifactory/api/docker/docker-local/v1/migrate" -H
"Content-Type: application/json" -d
'{"targetRepo": "docker-local2", "dockerRepository": "jfrog/ubuntu"}'
```

Deletion and Cleanup

Artifactory natively supports removing tags and repositories and complies with the [Docker Hub Spec](#).

Deletion of Docker tags and repositories automatically cleans up any orphan layers that are left (layers not used by any other tag/repository).

Currently, the Docker client does not support DELETE commands, but deletion can be triggered manually using cURL. Here are some examples:

Removing repositories and tags

```
//Removing the "jfrog/ubuntu" repository
curl -uadmin:password -X DELETE
"https://artprod.company.com/v1/repositories/jfrog/ubuntu"

//Removing the "12.04" tag from the "jfrog/ubuntu" repository
curl -uadmin:password -X DELETE
"https://artprod.company.com/v1/repositories/jfrog/ubuntu/tags/12.04"
```

Empty Directories

Any empty directories that are left following removal of a repository or tag will automatically be removed during the next folder pruning job (which occurs every 5 minutes by default).

Advanced Topics

Using a Self-signed SSL Certificate

From Docker version 1.3.1, you can use self-signed SSL certificates with `docker push/pull` commands, however for this to work, you need to specify the `--insecure-registry` daemon flag for each insecure registry.

For full details please refer to the [Docker documentation](#).

For example, if you are running Docker as a service, edit the `/etc/default/docker` file, and append the `--insecure-registry` flag with your registry URL to the `DOCKER_OPTS` variable as in the following example:

Edit the DOCKER_OPTS variable

```
DOCKER_OPTS="-H unix:///var/run/docker.sock --insecure-registry  
artprod.company.com"
```

For this to take effect, you need to restart the Docker service.

If you are using **Boot2Docker**, please refer to the **Boot2Docker** documentation for [Insecure Registry](#).

If you do not make the required modifications to the `--insecure-registry` daemon flag, you should get the following error:

Error message

```
Error: Invalid registry endpoint https://artprod.company.com/v1/: Get  
https://artprod.company.com/v1/_ping: x509: certificate signed by unknown  
authority.
```

Using previous versions of Docker

In order to use self-signed SSL certificates with previous versions of Docker, you need to manually install the certificate into the OS of each machine running the Docker client (see [Issue 2687](#)).

Alternative Proxy Servers

In addition to NGINX, you can setup Artifactory to work with Docker using Apache.

Apache Configuration

The sample configuration below configures SSL on port 443 and a server name of `artprod.company.com`.

▼ [Apache config for docker V1](#)

```
<VirtualHost *:443>
    ServerName artprod.company.com

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    SSLEngine on
    SSLCertificateFile/etc/ssl/certs/artprod.company.com.pem
    SSLCertificateKeyFile /etc/ssl/private/artprod.company.com.key

    ProxyRequests off
    ProxyPreserveHost on

    ProxyPass          /
    http://artprod.company.com:8080/artifactory/api/docker/docker-local/
    ProxyPassReverse  /
    http://artprod.company.com:8080/artifactory/api/docker/docker-local/
</VirtualHost>
```

Port Bindings

If you want to use multiple repositories, you need to copy the [NGINX configuration](#) and bind different ports to each local repository in Artifactory.

When binding a port other than 443, note that the configuration for the proxy header must be appended with the port number on the `proxy_set_header` line.

For example, for a server running on port 444 you should write `proxy_set_header Host $host:444`.

Docker Repository Path and Domain

When accessing a Docker repository through Artifactory, the repository URL must be prefixed with **api/docker** in the path.

You can copy the full URL from the UI using **Set Me Up** when the repository is selected in the Tree Browser.

For example, if you are using Artifactory standalone or as a local service, you would access your Docker repositories using the following URL:

```
http://localhost:8081/artifactory/api/docker/<repository key>
```

Also, the domain of your Docker repository must be expressed as an explicit IP address. The only exception is when working locally, you can use the `localhost` domain name as the proxy pass.

Support Matrix

Please refer to the [support matrix](#) under Docker Repositories.

Git LFS Repositories

Overview

From version 3.9, Artifactory supports [Git Large File Storage \(LFS\)](#) repositories on top of Artifactory's [existing support](#) for advanced artifact management.

Artifactory support for Git LFS provides you with a fully functional LFS server that works with the Git LFS client.

LFS blobs from your Git repository can be pushed and maintained in Artifactory offering the following benefits:

- **Performance:**
With Artifactory's file storage on your local or corporate network, file download times may be significantly reduced. When considering the number of files that may be needed for a build, this can drastically reduce your build time and streamline your workflow.
- **Reliable and consistent access to binaries:**
With Artifactory as your LFS repository, all the resources you need for development and build are stored on your own local or corporate network and storage. This keeps you independent of the external network or any 3rd party services.
- **Share binary assets with remote Git LFS repositories**
Share your video, audio, image files and any other binary asset between teams across your organization by proxying Git LFS repositories on other Artifactory instances or on GitHub.
- **Upload and download binary assets using a single URL**
Use a virtual Git LFS repository as both a source and a target for binary assets. By wrapping local and remote repositories, and defining a deploy target in a virtual Git LFS repository, your Git LFS client only needs to be exposed to that single virtual repository for all your work with binary assets.
- **Security and access control:**
Artifactory lets you define which users or groups of users can access your LFS repositories with a full set of permissions you can configure. You can control where developers can deploy binary assets to, whether they can delete assets and more. And if it's access to your servers that you're concerned about, Artifactory provides full integration with the most common access protocols such as LDAP, SAML, Crowd and others.
- **One solution for all binaries:**
Once you are using Artifactory to store media assets there is no need to use a 3rd party LFS provider. Artifactory can now handle those along with all the other binaries it already manages for you.

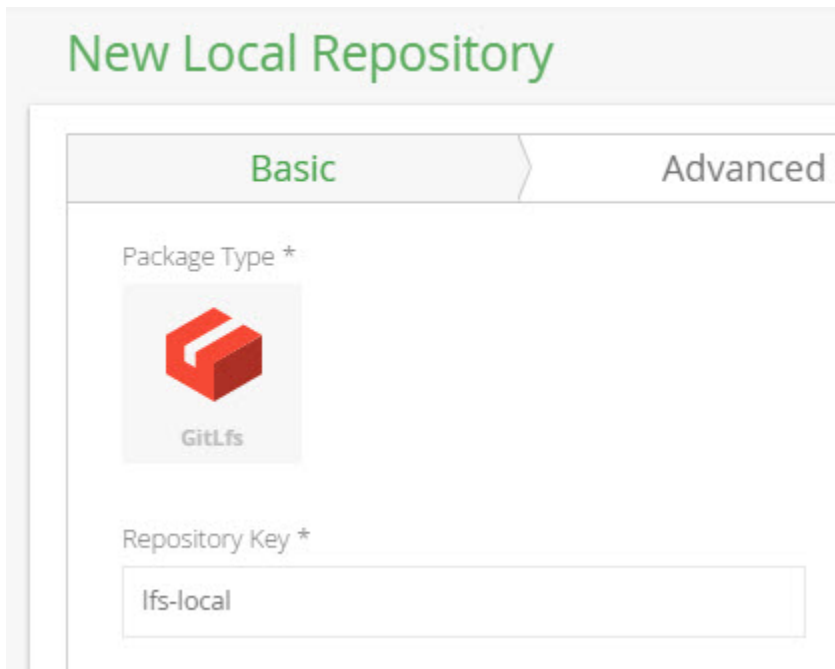
Page Contents

- [Overview](#)
- [Configuration](#)
 - [Local Repositories](#)
 - [Remote Repositories](#)
 - [Virtual Repositories](#)
 - [Setting Up the Git LFS Client to Point to Artifactory](#)
- [Working with Artifactory without Anonymous Access](#)
- [Authenticating with SSH](#)
- [Metadata](#)
- [Storage](#)
- [5-Minute Setup](#)

Configuration

Local Repositories

To create a Git LFS local repository and enable calculation of LFS package metadata set **GitLfs** as the **Package Type**.



New Local Repository

Basic | Advanced

Package Type *

GitLfs

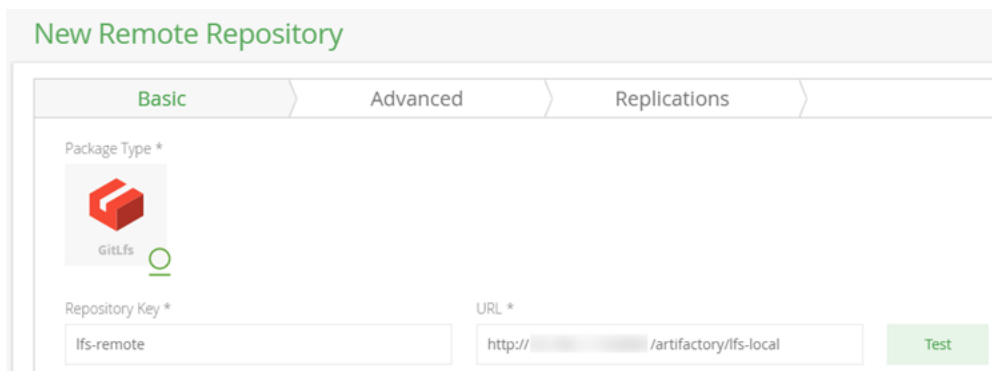
Repository Key *

lfs-local

Remote Repositories

You can create a Git LFS [remote repository](#) to proxy LFS repositories on GitHub, or Git LFS local repositories on other Artifactory instances. If you are proxying a Git LFS local repository on another instance of Artifactory, you can enjoy all the features of a [smart remote repository](#).

To define a Git LFS remote repository, create a new remote repository, set its Package Type to be **Git LFS**, and set the URL of the repository you want to proxy.



New Remote Repository

Basic | Advanced | Replications

Package Type *

GitLfs

Repository Key *

lfs-remote

URL *

http://[redacted]/artifactory/lfs-local

Test

Virtual Repositories

A Virtual Repository defined in Artifactory aggregates packages from both local and remote repositories.

This allows you to access both locally hosted binary assets and remote proxied git LFS repositories from a single URL defined for the virtual repository.

To create a Git LFS virtual repository set **Git LFS** to be its Package Type, and select the underlying local and remote Git LFS repositories to include under the **Repositories** section.

Make sure you also set the **Default Deployment Repository** so you can both download from and upload to this repository.

Repositories

Filter...

Available Repositories

Selected Repositories

- ifs-local
- ifs-remote

Included Repositories

- ifs-local
- ifs-remote

Default Deployment Repository

ifs-local

Setting Up the Git LFS Client to Point to Artifactory

In order for your client to upload and download LFS blobs from artifactory the `[lfs]` clause should be added to the `.lfsconfig` file of your Git repository in the following format.

```
.lfsconfig
[lfs]
  url = "https://<artifactory server path>/api/lfs/<LFS repo key>"

For example:
[lfs]
  url = "https://localhost:8080/artifactory/api/lfs/lfs-local"
```

You can also set different LFS endpoints for different remotes on your repo (as supported by the Git LFS client), for example:

```
.git/config different lfs url for remotes
[remote "origin"]
  url = https://...
  fetch = +refs/heads/*:refs/remotes/origin/*
  lfsurl = "http://localhost:8081/artifactory/api/lfs/lfs-local"
```

Copy these clauses using Set Me Up

If you select your GitLFS repository in the Tree Browser and click **Set Me Up**, Artifactory will display these clauses in a dialog from which you can simply copy and paste them.

Set Me Up

Tool

GitLfs

Repository

lfs-local

Resolve

In order for your client to upload and download LFS blobs from artifactory the [lfs] clause should be added to the .gitconfig file of your Git repository in the following format.

```
1 [lfs]
2 url = "http://10.100.1.110:8081/artifactory/api/lfs/lfs-local"
```

You can also set different LFS endpoints for different remotes on your repo (as supported by the Git LFS client), for example:

```
1 [remote "origin"]
2 url = <URL>
3 fetch = +refs/heads/*:refs/remotes/origin/*
4 lfsurl = "http://10.100.1.110:8081/artifactory/api/lfs/lfs-local"
```

Working with Proxies and HTTPS

When using HTTPS (i.e. behind a proxy) with a self signed certificate your configuration might also require you to add the following:

.gitconfig http section

```
[http]
  sslverify = false
```

Always consult your System Administrator before bypassing secure protocols in this way.

When running Artifactory behind a proxy, defining a **base url** is usually required (depending on configuration) due to the operation of the

Git LFS client which expects to receive redirect urls to the exact upload \ download location of blobs.

LFS repositories must be prefixed with api/lfs in the path

When accessing a Git LFS repository through Artifactory, the repository URL must be prefixed with **api/lfs** in the path, **except when configuring replication**.

For example, if you are using Artifactory standalone or as a local service, you would access your LFS repositories using the following URL:

```
http://localhost:8081/artifactory/api/lfs/<repository key>
```

Or, if you are using SaaS the URL would be:

```
https://<server name>.jfrog.io/<server name>/api/lfs/<repository key>
```

When configuring replication, reference the repository's browsable url i.e.;

```
http://localhost:8081/artifactory/<repository key>
```

Working with Artifactory without Anonymous Access

By default, Artifactory allows anonymous access to Git LFS repositories. This is defined in the **Admin** module under **Security | General** . For details please refer to [Allow Anonymous Access](#).

If you want to be able to trace how users interact with your repositories you need to uncheck the [Allow Anonymous Access](#) setting. This means that users will be required to enter their username and password.

The Git LFS client will ask for credentials for the Artifactory LFS repo when accessing it - if anonymous access is allowed you can just enter blank credentials, otherwise you should enter your Artifactory user name and password (not your Git one).

To make the authentication process automatic you can use [Git Credential Helpers](#) to store these for you, and have the Git LFS client authenticate automatically.

Git stores credentials in plain text by default

You should take extra measures to secure your username and password when using Git credential helpers

Authenticating with SSH

From version 4.4, Artifactory supports authenticating your Git LFS client via SSH.

To authenticate yourself via SSH when using the Git LFS client, execute the following steps:

- Make sure Artifactory is properly configured for SSH as described in [Configuring Server Authentication](#).

SSH Server Settings

Enable SSH Authentication

Port *

Custom URL Base [?](#)

- Upload your SSH Public Key in the SSH section of your user profile as described in [Configuring User Authentication](#).
- Configure the Git LFS client as follows:
 - Update the *known_hosts* file with the Artifactory server public key. This file is located under *~/.ssh/known_hosts* (and there is also a system-wide file under */etc/ssh/known_hosts*). This should take the following format:
[<server_custom_base_URL>]:<server_port> <content of the Artifactory server public ssh key>
For example,

```
[myartifactory.company.com]:1339 ssh-rsa  
AAAAB3Nza...PC0GuTJT9TlaYD user@domain.com
```

- Update your *.lfsconfig* file at the repository level (not the global level) as follows:
ssh://\$USERNAME@\$HOST:\$PORT/artifactory/<repoKey>
For example,

```
url =  
"ssh://git@myartifactory.company.com:1339/artifactory/lfs-local"
```

Artifactory Online Dedicated Server

If you are using a dedicated server on Artifactory SaaS and wish to authenticate via SSH, please contact support@jfrog.com.

Metadata

As the Git LFS client supplies only limited data about the blob being uploaded (only its sha256 checksum, or 'OID', and its size) Artifactory does not store or process any metadata for LFS blobs.

You can set properties on the blobs for your own convenience but this requires extra logic to infer a sha256-named file stored in artifactory from the actual pointer stored in your Git repository.

Storage

Artifactory stores LFS blobs in a manner similar to the Git LFS client, using the provided sha256 checksum.

Git LFS blobs will be stored under a path such as `<lfs_repo>/objects/ad/1b/ad1b8d6e1cafd33e941a5de462ca7edfa8818a70c79F`
`ea68e5ed53dec414c4`

Where **ad** and **1b** are the 1st and 2nd, and the 3rd and 4th characters in the blob's name respectively.

Git LFS behavior when download from the LFS endpoint fails

The Git LFS client will download the pointer file it created in your remote Git repository if downloading the blob from the LFS endpoint failed (i.e. wrong credentials, network error etc.).

This will cause the actual file in your local repo to be substituted with the pointer created by the LFS client **with the same name** and lead to any number of problems this behavior can cause.

This is a limitation of the LFS client tracked by [issue 89](#).

5-Minute Setup

Visit our Knowledge Base and learn how to [setup Git LFS to work with Artifactory in 5 minutes](#).

Npm Registry

Overview

Artifactory provides full support for managing npm packages and ensures optimal and reliable access to *npmjs.org*. Aggregating multiple npm registries under a virtual repository Artifactory provides access to all your npm packages through a single URL for both upload and download.

As a fully-fledged npm registry on top of its capabilities for [advanced artifact management](#), Artifactory's support for [npm](#) provides:

1. The ability to provision npm packages from Artifactory to the npm command line tool from all repository types
2. Calculation of Metadata for npm packages hosted in Artifactory's local repositories
3. Access to remote npm registries (such as <https://registry.npmjs.org>) through [Remote Repositories](#) which provide the usual proxy and caching functionality
4. The ability to access multiple npm registries from a single URL by aggregating them under a [Virtual Repository](#). This overcomes the limitation of the npm client which can only access a single registry at a time.
5. Compatibility with the [npm command line tool](#) to deploy and remove packages and more.
6. Support for [flexible npm repository layouts](#) that allow you to organize your npm packages and assign access privileges according to projects or development teams.

Npm version support

Artifactory supports NPM version 1.4.3 and above.

Configuration

Local Npm Registry

To enable calculation of npm package metadata in local repositories so they are, in effect, npm registries, set the **Package Type** to **npm** when you create the repository:

New Local Repository

Basic

Package Type *



Page Contents

- Overview
- Configuration
 - Local Npm Registry
- Repository Layout
 - Remote Npm Registry
 - Virtual Npm Registry
 - Advanced Configuration
- Using the Npm Command Line
 - Setting the Default Registry
 - Authenticating the npm Client
 - Using npm login
 - Using Basic Authentication
 - Resolving npm Packages
- Npm Publish (Deploying Packages)
 - Setting Your Credentials
 - Deploying Your Packages
- Specifying the Latest Version
- Working with Artifactory without Anonymous Access
- Using OAuth Credentials
- Npm Search
- Cleaning Up the Local Npm Cache
- Npm Scope Packages

- Configuring the npm Client for a Scope Registry
 - Using Login Credentials
- Automatically Rewriting External Dependencies
 - Rewriting Workflow
- Viewing Individual Npm Package Information
- Watch the Screencast

Repository Layout

Artifactory allows you to define any layout for your npm registries. In order to upload packages according to your custom layout, you need to package your npm files using `npm pack`.

This creates the `.tgz` file for your package which you can then upload to any path within your local npm repository.

Remote Npm Registry

A **Remote Repository** defined in Artifactory serves as a caching proxy for a registry managed at a remote URL such as <https://registry.npmjs.org>.

Artifacts (such as `tgz` files) requested from a remote repository are cached on demand. You can remove downloaded artifacts from the remote repository cache, however, you can not manually deploy artifacts to a remote npm registry.

To define a remote repository to proxy a remote npm registry follow the steps below:

1. In the **Admin** module, under **Repositories | Remote**, click "New".
2. In the New Repository dialog, set the **Package Type** to **npm**, set the **Repository Key** value, and specify the URL to the remote registry in the **URL** field as displayed below

The screenshot shows the 'New Remote Repository' configuration window. It has three tabs: 'Basic', 'Advanced', and 'Replications'. The 'Basic' tab is selected. Under 'Package Type *', there is a dropdown menu with 'npm' selected. Below that, 'Repository Key *' has a text input field containing 'npm-remote'. To the right, 'URL *' has a text input field containing 'https://registry.npmjs.org'. A green 'Test' button is located to the right of the URL field.

3. Click "Save & Finish"

Virtual Npm Registry

A **Virtual Repository** defined in Artifactory aggregates packages from both local and remote repositories.

This allows you to access both locally hosted npm packages and remote proxied npm registries from a single URL defined for the virtual repository.

To define a virtual npm registry, create a **virtual repository**, set the **Package Type** to be **npm**, and select the underlying local and remote npm registries to include in the **Basic** settings tab.

Repositories

Filter...

Available Repositories

- mytestnpm-local
- npm

Selected Repositories

- npm-local
- npm-remote

Included Repositories

- npm-local
- npm-remote

Click "Save & Finish" to create the repository.

Advanced Configuration

Edit npm-virtual Repository

Basic > **Advanced**

Artifactory Requests Can Retrieve Remote Artifacts ?

External Dependency Rewrite

Enable Dependency Rewrite

Remote Repository For Cache

npm-remote

Patterns Whitelist ?

New Pattern Add

/github.com/

The fields under **External Dependency Rewrite** are connected to [automatically rewriting external dependencies](#) for npm packages that need them.

Enable Dependency Rewrite	When checked, automatically rewriting external dependencies is enabled.
----------------------------------	---

Remote Repository For Cache	The remote repository aggregated by this virtual repository in which the external dependency will be cached.
Patterns Whitelist	<p>A white list of Ant-style path expressions that specify where external dependencies may be downloaded from. By default, this is set to <code>**</code> which means that dependencies may be downloaded from any external source.</p> <p>For example, if you wish to limit external dependencies to only be downloaded from <code>github.com</code>, you should add <code>**github.com**</code> (and remove the default <code>**</code> expression).</p>

Using the Npm Command Line

Npm repositories must be prefixed with `api/npm` in the path

When accessing an npm repository through Artifactory, the repository URL must be prefixed with `api/npm` in the path. This applies to all npm commands including `npm install` and `npm publish`.

For example, if you are using Artifactory standalone or as a local service, you would access your npm repositories using the following URL:

```
http://localhost:8081/artifactory/api/npm/<repository key>
```

Or, if you are using Artifactory SaaS the URL would be:

```
https://<server name>.jfrog.io/<server name>/api/npm/<repository key>
```

To use the npm command line you need to make sure npm is installed. Npm is included as an integral part of recent versions of [Node.js](#).

Please refer to [Installing Node.js via package manager](#) on GitHub or the [npm README page](#).

Once you have created your npm repository, you can select it in the Tree Browser and click **Set Me Up** to get code snippets you can use to change your npm registry URL, deploy and resolve packages using the npm command line tool.

Setting the Default Registry

For your npm command line client to work with Artifactory, you first need to set the default npm registry with an Artifactory npm repository using the following command (the example below uses a repository called `npm-repo`):

Replacing the default registry

```
npm config set registry
http://<ARTIFACTORY_SERVER_DOMAIN>:8081/artifactory/api/npm/npm-repo
```

For scoped packages, use the following command:

```
npm config set @<SCOPE>:registry
http://<ARTIFACTORY_SERVER_DOMAIN>:8081/artifactory/api/npm/npm-repo
```

We recommend referencing a [Virtual Repository](#) URL as a registry. This gives you the flexibility to reconfigure and aggregate other external sources and local repositories of npm packages you deployed.

Note that if you do this, you need to use the `--registry` parameter to specify the local repository into which you are publishing your package when using the `npm publish` command.

Authenticating the npm Client

Once you have set the default registry, you need to authenticate the npm client to Artifactory in one of two ways: using the `npm login` command or using basic authentication.

Using npm login

Authentication using `npm login` was introduced in version 5.4.

Run the following command in your npm client. When prompted, provide your Artifactory login credentials:

```
npm login
```

Upon running this command, Artifactory creates a [non-expirable access token](#) which the client uses for authentication against Artifactory for subsequent `npm install` and `npm publish` actions.

If the token is removed from Artifactory, the client will have to login again to receive a new token.

Using Basic Authentication

To support basic authentication you need to edit your `.npmrc` file and enter the following:

- Your Artifactory username and password (formatted `username:password`) as [Base64](#) encoded strings
- Your email address (`npm publish` will not work if your email is not specified in `.npmrc`)
- You need to set `always-auth = true`

.npmrc file location

Windows: `%userprofile%\ .npmrc`

Linux: `~/.npmrc`

Getting .npmrc entries directly from Artifactory

You can use the following command to get these strings directly from Artifactory:

```
$ curl -uadmin:<CREDENTIAL>  
http://<ARTIFACTORY_SERVER_DOMAIN>:8081/artifactory/api/npm/auth
```

Where `<CREDENTIAL>` is your Artifactory password or [API Key](#)

Here is an example of the response:

```
_auth = YWRtaW46e0RFU2VkZX1uOFRaaXh1Y0t3bHN4c2RCTVIwNjF3PT0=  
email = myemail@email.com  
always-auth = true
```

If, in addition, you are also working with scoped packages, you also need to run the following command:

```
curl -uadmin:<CREDENTIAL>  
http://<ARTIFACTORY_SERVER_DOMAIN>:8081/artifactory/api/npm/npm-repo/auth/  
<SCOPE>
```

Where `<CREDENTIAL>` is your Artifactory password or [API Key](#)

Paste the response to this command in the `~/.npmrc` file on your machine (in Windows, `%USERPROFILE%\ .npmrc`).

Resolving npm Packages

Once the npm command line tool is configured, every `npm install` command will fetch packages from the npm repository specified above. For example:

```
$ npm install request
npm http GET http://localhost:8081/artifactory/api/npm/npm-repo/request
npm http 200 http://localhost:8081/artifactory/api/npm/npm-repo/request
npm http GET
http://localhost:8081/artifactory/api/npm/npm-repo/request/-/request-2.33.0.tgz
npm http 200
http://localhost:8081/artifactory/api/npm/npm-repo/request/-/request-2.33.0.tgz
```

Npm Publish (Deploying Packages)

Setting Your Credentials

The npm command line tool requires that sensitive operations, such as `publish`, are authenticated as described under [Authenticating the npm Client](#) above.

Deploying Your Packages

There are two ways to deploy packages to a local repository:

- Edit your `package.json` file and add a `publishConfig` section to a local repository:
`"publishConfig":{"registry":"http://localhost:8081/artifactory/api/npm/npm-local"}`
- Provide a local repository to the `npm publish` command:
`npm publish --registry http://localhost:8081/artifactory/api/npm/npm-local`

Specifying the Latest Version

By default, the "latest" version of a package in an NPM registry in Artifactory is the one with the highest [SemVer](#) version number. You can override this behavior so that the most recently uploaded package is returned by Artifactory as the "latest" version. To do so, in Artifactory's `system.properties` file, add or set:

```
artifactory.npm.tag.tagLatestByPublish = true
```

Working with Artifactory without Anonymous Access

By default, Artifactory allows anonymous access to npm repositories. This is defined in the **Admin** module under **Security | General**. For details please refer to [Allow Anonymous Access](#).

If you want to be able to trace how users interact with your repositories you need to uncheck the [Allow Anonymous Access](#) setting. This means that users will be required to enter their username and password as described in [Setting Your Credentials](#) above.

Using OAuth Credentials

Artifactory uses GitHub Enterprise as its [default OAuth provider](#). If you have an account, you may use your GitHub Enterprise login details to be authenticated when using `npm login`.

Npm Search

Artifactory supports a variety of ways to search of artifacts. For details please refer to [Searching Artifacts](#).

Artifactory also supports `npm search [search terms ...]`, however, packages may not be available immediately after being published for

the following reasons:

When publishing a package to a local repository, Artifactory calculates the search index asynchronously and will wait for a "quiet period" to lapse before indexing the newly published package.

Since a virtual repository may contain local repositories, a newly published package may not be available immediately for the same reason.

You can specify the indexing "quiet period" (time since the package was published) by setting the following system properties (in `$ARTIFACTORY_HOME/etc/artifactory.system.properties`).

```
artifactory.npm.index.quietPeriodSecs=60
artifactory.npm.index.cycleSecs=60
```

In the case of remote repositories, a new package will only be found once Artifactory checks for it according to the [Retrieval Cache Period](#) setting.

Artifactory annotates each deployed or cached npm package with two properties: `npm.name` and `npm.version`

You can use [Property Search](#) to search for npm packages according to their name or version.

Cleaning Up the Local Npm Cache

The npm client saves caches of packages that were downloaded, as well as the JSON metadata responses (named `.cache.json`).

The JSON metadata cache files contain URLs which the npm client uses to communicate with the server, as well as other ETag elements sent by previous requests.

We recommend removing the npm caches (both packages and metadata responses) before using Artifactory for the first time. This is to ensure that your caches only contain elements that are due to requests from Artifactory and not directly from <https://registry.npmjs.org>.

The default cache directory on Windows is `%APPDATA%\npm-cache` while on Linux it is `~/.npm`.

Npm Scope Packages

Artifactory fully supports [npm scope packages](#). The support is transparent to the user and does not require any different usage of the npm client.

Npm 'slash' character encoding

By default, the npm client encodes slash characters (`/`) to their ASCII representation (`"%2f"`) before communicating with the npm registry. If you are running Tomcat as your HTTP container (the default for Artifactory), this generates an "HTTP 400" error since Tomcat does not allow encoded slashes by default. In order to work with npm scoped packages, you can override this default behavior by defining the following property in the `catalina.properties` file of your Tomcat:

```
org.apache.tomcat.util.buf.UDECODER.ALLOW_ENCODED_SLASH=true
```

Note that since Artifactory **version 4.4.3**, the bundled Tomcat is configured by default to enable encoded slashes. If you are using a previous version you will need to adjust the Tomcat property above.

URL decoding and reverse proxy

If Artifactory is running behind a reverse proxy, make sure to disable URL decoding on the proxy itself in order to work with npm scope packages.

For Apache, add the `"AllowEncodedSlashes NoDecode"` directive inside the `<VirtualHost *:xxx>` block.

Configuring the npm Client for a Scope Registry

Using Login Credentials

Scopes can be associated with a separate registry. This allows you to seamlessly use a mix of packages from the public npm registry and one or more private registries.

For example, you can associate the scope `@jfrog` with the registry `http://localhost:8081/artifactory/api/npm/npm-local/` by manually altering your `~/.npmrc` file and adding the following configuration:

```
@jfrog:registry=http://localhost:8081/artifactory/api/npm/npm-local/  
//localhost:8081/artifactory/api/npm/npm-local/:_password=cGFzc3dvcmQ=  
//localhost:8081/artifactory/api/npm/npm-local/:username=admin  
//localhost:8081/artifactory/api/npm/npm-local/:email=myemail@email.com  
//localhost:8081/artifactory/api/npm/npm-local/:always-auth=true
```

Getting .npmrc entries directly from Artifactory

From Artifactory 3.5.3, you can use the following command to get these strings directly from Artifactory:

```
$ curl -uadmin:password "http://localhost:8081/artifactory/api/npm/npm-local/auth/jfrog"  
@jfrog:registry=http://localhost:8081/artifactory/api/npm/npm-local/  
//localhost:8081/artifactory/api/npm/npm-local/:_password=QVA1N05OaHZTMnM5Qk02RkR5RjNBVmf4TVF1  
//localhost:8081/artifactory/api/npm/npm-local/:username=admin  
//localhost:8081/artifactory/api/npm/npm-local/:email=admin@jfrog.com  
//localhost:8081/artifactory/api/npm/npm-local/:always-auth=true
```

User email is required

When using scope authentication, npm expects a valid email address. Please make sure you have included your email address in your Artifactory user profile.

The password is just a base64 encoding of your Artifactory password, the same way used by the [old authentication configuration](#).

Recommend npm command line tool version 2.1.9 and later.

While npm scope packages have been available since version 2.0 of the npm command line tool, we highly recommend using npm scope packages with Artifactory only from version 2.1.9 of the npm command line tool.

Automatically Rewriting External Dependencies

Packages requested by the Npm client frequently use external dependencies as defined in the packages' `package.json` file. These dependencies may, in turn, need additional dependencies. Therefore, when downloading an Npm package, you may not have full visibility into the full set of dependencies that your original package needs (whether directly or transitively). As a result, you are at risk of downloading malicious dependencies from unknown external resources. To manage this risk, and maintain the best practice of consuming external packages through Artifactory, you may specify a "safe" whitelist from which dependencies may be downloaded, cached in Artifactory and configure to rewrite the dependencies so that the Npm client accesses dependencies through a virtual repository as follows:

- Check **Enable Dependency Rewrite** in the Npm virtual repository [advanced configuration](#).
- Specify a whitelist patterns of external resources from which dependencies may be downloaded.
- Specify the remote repository in which those dependencies should be cached.
It is preferable to configure a dedicated remote repository for that purpose so it is easier to maintain.

In the example below the external dependencies will be cached in "npm" remote repository and only package from `https://github.com/jfrog/gdev` are allowed to be cached.

New Virtual Repository

Basic > **Advanced**

Artifactory Requests Can Retrieve Remote Artifacts ?

External Dependency Rewrite

Enable Dependency Rewrite

Remote Repository For Cache

npm

Patterns Whitelist ?

New Pattern +

/github.com/jfrogdev/

Artifactory supports all possible shorthand resolvers including the following:

```
git+ssh://user@hostname:project.git#commit-ish
git+ssh://user@hostname/project.git#commit-ish
git+https://git@github.com/<user>/<filename>.git
```

Rewriting Workflow

1. When downloading an Npm package, Artifactory analyzes the list of dependencies needed by the package.
2. If any of the dependencies are hosted on external resources (e.g. on *github.com*), and those resources are specified in the white list,
 - a. Artifactory will download the dependency from the external resource.
 - b. Artifactory will cache the dependency in the remote repository configured to cache the external dependency.
 - c. Artifactory will then modify the dependency's entry in the package's *package.json* file indicating its new location in the Artifactory remote repository cache before returning it to the Npm client.
3. Consequently, every time the Npm client needs to access the dependency, it will be provisioned from its new location in the Artifactory remote repository cache.

Viewing Individual Npm Package Information

Artifactory lets you view selected metadata of an npm package directly from the UI.

In the **Tree Browser**, drill down to select the tgz file you want to inspect. The metadata is displayed in the **Npm Info** tab.

Package Info

Name:	request
Version:	2.31.0
Description:	Simplified HTTP request client.
Repository:	

Dependencies

< page 1 of 1 >

Name	Version
qs	~0.6.0
json-stringify-safe	~5.0.0
forever-agent	~0.5.0
node-uuid	~1.4.0
mime	~1.2.9

Watch the Screencast

NuGet Repositories

Overview

From version 2.5, Artifactory provides complete support for NuGet repositories on top of Artifactory's existing support for advanced artifact management.

Artifactory support for NuGet provides:

1. The ability to provision NuGet packages from Artifactory to NuGet clients from all repository types
2. Metadata calculation for NuGet packages hosted in Artifactory's local repositories
3. The ability to define proxies and caches to access Remote NuGet repositories
4. Multiple NuGet repository aggregation through virtual repositories
5. APIs to deploy or remove packages that are compatible with NuGet Package Manager Visual Studio extension and the NuGet Command Line Tool
6. Debugging NuGet packages directly using Artifactory as a Microsoft Symbol Server

Metadata updates

NuGet metadata is automatically calculated and updated when adding, removing, copying or moving NuGet packages. The calculation is only invoked after a package-related action is completed.

It is asynchronous and its performance depends on the overall system load, therefore it may sometimes take up to 30 seconds to complete.

You can also invoke metadata calculation on the entire repository by selecting "Reindex Packages".

Page Contents

- Overview
- Configuration
 - Local Repositories
 - Local

Repository Layout

- Publishing to a Local Repository
 - Remote Repositories
 - Virtual Repositories
- Accessing NuGet Repositories from Visual Studio
- Using the NuGet Command Line
- NuGet API Key Authentication
- Anonymous Access to NuGet Repositories
 - Working Without Anonymous Access
 - Allowing Anonymous Access
- Viewing Individual NuGet Package Information
- Watch the Screencast

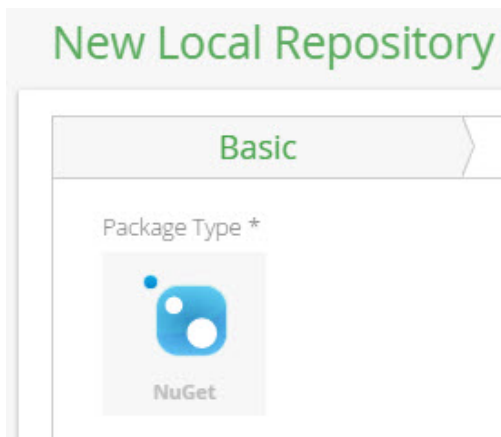
Read More

- Microsoft Symbol Server

Configuration

Local Repositories

To create a local repository for which Artifactory will calculate NuGet package metadata, set **NuGet** to be the **Package Type**.



Local Repository Layout

To support a more manageable repository layout, you may store NuGet packages inside folders that correspond to the package structure.

Artifactory will find your packages by performing a property search so the folder hierarchy does not impact performance.

To use a hierarchical layout for your repository you should define a [Custom Layout](#). This way, different maintenance features like [Version Cleanup](#) will work correctly with NuGet packages.

Placing packages to match your repository layout

Defining a [Custom Layout](#) for your repository does not force you to place your packages in the corresponding structure, however it is recommended to do so since it allows Artifactory to perform different maintenance tasks such as [Version Cleanup](#) automatically.

It is up to the developer to correctly deploy packages into the corresponding folder. From NuGet 2.5 you can push packages into a folder source as follows:

```
nuget push mypackage.1.0.0.nupkg -Source  
http://10.0.0.14:8081/artifactory/api/nuget/nuget-local/path/to/folder
```

Below is an example of a [Custom Layout](#) named `nuget-default`.

Edit nuget-default Repository Layout

Repository Layout Settings

Layout Name *

nuget-default

Artifact Path Pattern *

[orgPath]/[module]/[module].[baseRev](-[fileIntegRev]).r

Distinctive Descriptor Path Pattern

Folder Integration Revision RegExp *

.*

File Integration Revision RegExp *

.*

Test Artifact Path Resolution

Test Path

NHibernate/NHibernate/NHibernate.3.3.3-CR1.nupkg

Test

Result

Organization: NHibernate

Module: NHibernate

Base Revision: 3.3.3

Folder Integration Revision:

File Integration Revision: CR1

Classifier:

Extension:

Type:

In this example, the three fields that are mandatory for module identification are:

- Organization = "orgPath"
- Module = "module"
- Base Revision ("baseRev") is not a part of the layout hierarchy in this example, but it is included here as one of the required fields.

You can configure this Custom Layout as displayed in the image above, or simply copy the below code snippet into the relevant section in your Global Configuration Descriptor:

```

<repoLayout>
  <name>nuget-default</name>

  <artifactPathPattern>[orgPath]/[module]/[module].[baseRev](-[fileIntegRev])
  .[ext]</artifactPathPattern>

  <distinctiveDescriptorPathPattern>>false</distinctiveDescriptorPathPattern>
    <folderIntegrationRevisionRegExp>.*</folderIntegrationRevisionRegExp>
    <fileIntegrationRevisionRegExp>.*</fileIntegrationRevisionRegExp>
</repoLayout>

```

Since the package layout is in a corresponding folder hierarchy, the Artifactory Version Cleanup tool correctly detects previously installed versions.

The screenshot shows the 'Delete Versions' dialog box in the Artifactory interface. The dialog has a search filter 'Filter by Group ID or Version' and a table with two rows. The table has columns for 'Group ID', 'Version', and 'Directories Count'. The first row shows 'Microsoft' with version 'Microsoft.AspNet.Mvc.5.0.0' and a count of 1. The second row shows 'Microsoft' with version 'Microsoft.AspNet.Mvc.5.2.0' and a count of 1. There are 'Cancel' and 'Delete Selected' buttons at the bottom right of the dialog.

Group ID	Version	Directories Count
Microsoft	Microsoft.AspNet.Mvc.5.0.0	1
Microsoft	Microsoft.AspNet.Mvc.5.2.0	1

Publishing to a Local Repository

When a NuGet repository is selected in the **Artifacts** module Tree Browser, click **Set Me Up** to display the code snippets you can use to configure Visual Studio or your NuGet client to use the selected repository to publish or resolve artifacts.

Tool

 NuGet

Repository

nuget-layout-local

Deploy

To support more manageable layouts, and additional features such as cleanup, NuGet repositories support custom layouts. You need to make sure that you push the package to a layout which matches the target repository layout.

```
1 nuget push <PACKAGE> -Source http://10.100.1.110:8081/artifactory/api/nuget/nuget-layout-local/<PATH_TO_FOLDER>
```

You can also add Artifactory repository as a gallery by running the following command

```
1 nuget sources Add -Name Artifactory -Source http://10.100.1.110:8081/artifactory/api/nuget/nuget-layout-local
```

This will enable you to use Artifactory with NuGet from command line for both pull and push.

Resolve

NuGet repositories must be prefixed with `api/nuget` in the path. When configuring Visual Studio to access a NuGet repository through Artifactory, the repository URL must be prefixed with `api/nuget` in the path.

```
1 http://10.100.1.110:8081/artifactory/api/nuget/nuget-layout-local
```

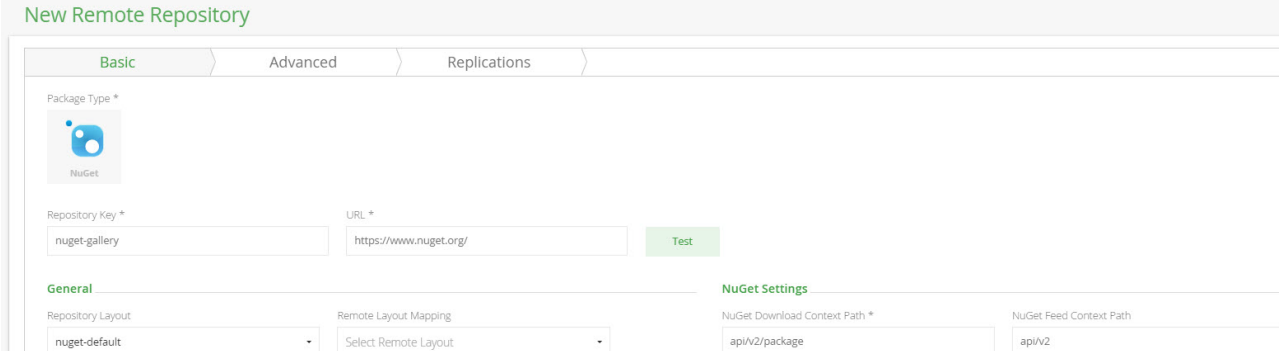
Remote Repositories

When working with remote NuGet repositories, your Artifactory configuration depends on how the remote repositories are set up.

Different NuGet server implementations may provide package resources on different paths, therefore the feed and download resource locations in Artifactory are customizable when proxying a remote NuGet repository.

Here are some examples:

- The **NuGet gallery** exposes its feed resource at <https://www.nuget.org/api/v2/Packages> and its download resource at <https://www.nuget.org/api/v2/package>. Therefore, to define this as a new repository you should set the repository **URL** to `https://www.nuget.org`, its **Feed Context Path** to `api/v2` and the **Download Context Path** to `api/v2/package`.



- The **NuGet . Server** module exposes its feed resource at <http://host:port/nuget/Packages> and its download resource at <http://host:port/api/v2/package>. To define this as a new repository you should set the repository **URL** to `http://host:port`, its **Feed Context Path** to `nuget` and its **Downlo**

ad Context Path to `api/v2/package`.

- Another Artifactory repository exposes its feed resource at `http://host:port/artifactory/api/nuget/repoKey/Packages` and its download resource at `http://host:port/artifactory/api/nuget/repoKey/Download`. To define this as a new repository you should set the repository URL to `http://host:port/artifactory/api/nuget/repoKey`, its **Feed Context Path** should be left empty and its **Download Context Path** to `Download`, like this:

NuGet Settings

NuGet Download Context Path * ?

NuGet Feed Context Path ?

Using a proxy server

If you are accessing NuGet Gallery through a proxy server you need to define the following two URLs in the proxy's white list:

1. `*.nuget.org`
2. `az320820.vo.msecnd.net` (our current CDN domain)

Virtual Repositories

A Virtual Repository defined in Artifactory aggregates packages from both local and remote repositories.

This allows you to access both locally hosted NuGet packages and remote proxied NuGet libraries from a single URL defined for the virtual repository.

To create a virtual Bower repository set **NuGet** to be its **Package Type**, and select the underlying local and remote NuGet repositories to include under the **Repositories** section.

Repositories

Filter...

Available Repositories

Selected Repositories

- nuget-layout-local X
- nuget-local X
- nuget X
- nuget-gallery X

Navigation: << < > >>

Included Repositories

nuget-layout-local
nuget-local
nuget
nuget-gallery

Accessing NuGet Repositories from Visual Studio

NuGet repositories must be prefixed with **api/nuget** in the path

When configuring Visual Studio to access a NuGet repository through Artifactory, the repository URL must be prefixed with **api/nuget** in the path.

For example, if you are using Artifactory standalone or as a local service, you would configure Visual Studio using the following URL:

```
http://localhost:8081/artifactory/api/nuget/<repository key>
```

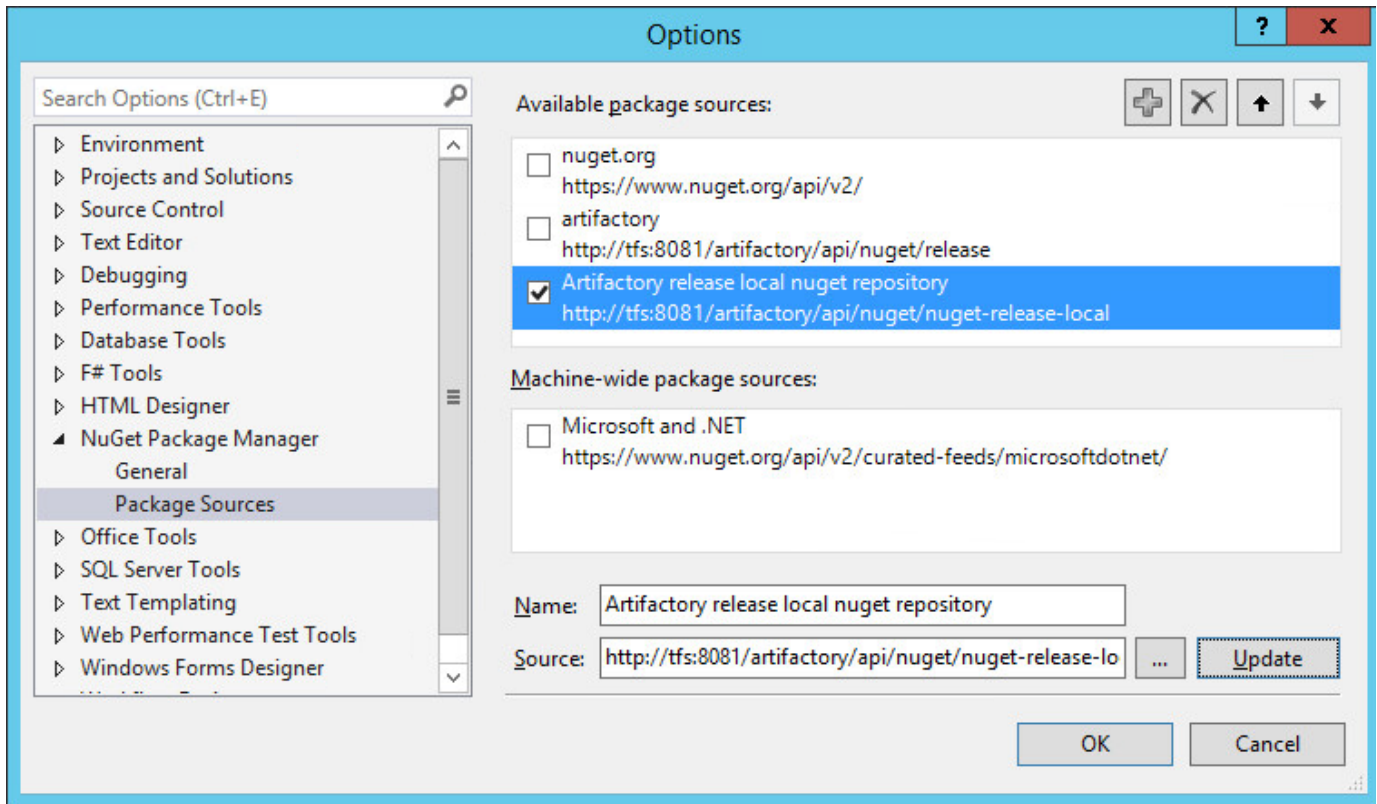
Or, if you are using Artifactory SaaS the URL would be:

```
https://<server name>.jfrog.io/<server name>/api/nuget/<repository key>
```

Artifactory exposes its NuGet resources via the REST API at the following URL: `http://localhost:8081/artifactory/api/nuget/<repository key>`.

This URL handles all NuGet related requests (search, download, upload, delete) and supports both V1 and V2 requests.

To configure the NuGet Visual Studio Extension to use Artifactory, check the corresponding repositories in the "Options" window: (You can access Options from the Tools menu).



Using the NuGet Command Line

NuGet repositories must be prefixed with `api/nuget` in the path

When using the NuGet command line to access a repository through Artifactory, the repository URL must be prefixed with `api/nuget` in the path. This applies to all NuGet commands including `nuget install` and `nuget push`.

For example, if you are using Artifactory standalone or as a local service, you would access your NuGet repositories using the following URL:

```
http://localhost:8081/artifactory/api/nuget/<repository key>
```

Or, if you are using Artifactory SaaS the URL would be:

```
https://<server name>.jfrog.io/<server name>/api/nuget/<repository key>
```

To use the NuGet Command Line tool:

1. Download NuGet.exe
2. Place it in a well known location in your file system such as `c:\utils`
3. Make sure that NuGet.exe is in your path

For complete information on how to use the NuGet Command Line tool please refer to the [NuGet Docs Command Line Reference](#).

First configure a new source URL pointing to Artifactory:

```
nuget sources Add -Name Artifactory -Source  
http://localhost:8081/artifactory/api/nuget/<repository key>
```

NuGet API Key Authentication

NuGet tools require that sensitive operations such as push and delete are authenticated with the server using an `apikey`. The API key you

should use is in the form of `username:password`, where the password can be either clear-text or [encrypted](#). Set your API key using the NuGet Command Line Interface:

```
nuget setapikey admin:password -Source Artifactory
```

Now you can perform operations against the newly added server. For example:

```
nuget list -Source Artifactory  
  
nuget install log4net -Source Artifactory
```

Anonymous Access to NuGet Repositories

By default, Artifactory allows anonymous access to NuGet repositories. This is defined under **Security | General Configuration**. For details please refer to [Allow Anonymous Access](#).

Working Without Anonymous Access

In order to be able to trace how users interact with your repositories we recommend that you uncheck the **Allow Anonymous Access** setting described above. This means that users will be required to enter their user name and password when using their NuGet clients.

You can configure your NuGet client to require a username and password using the following command:

```
nuget sources update -Name Artifactory -UserName admin -Password password
```

You can verify that your setting has taken effect by checking that the following segment appears in your `%APPDATA%\NuGet\NuGet.Config` file:

```
<packageSourceCredentials>  
  <Artifactory>  
    <add key="Username" value="admin" />  
    <add key="Password" value="...encrypted password..." />  
  </Artifactory>  
</packageSourceCredentials>
```

`NuGet.Config` file can also be placed in your project directory, for further information please refer to [NuGet Configuration File](#)

Allowing Anonymous Access

Artifactory supports NuGet repositories with **Allow Anonymous Access** enabled.

When **Allow Anonymous Access** is enabled, Artifactory will not query the NuGet client for authentication parameters by default, so you need to indicate to Artifactory to request authentication parameters in a different way.

You can override the default behavior by setting the **Force Authentication** checkbox in the New or Edit Repository dialog.

NuGet Settings

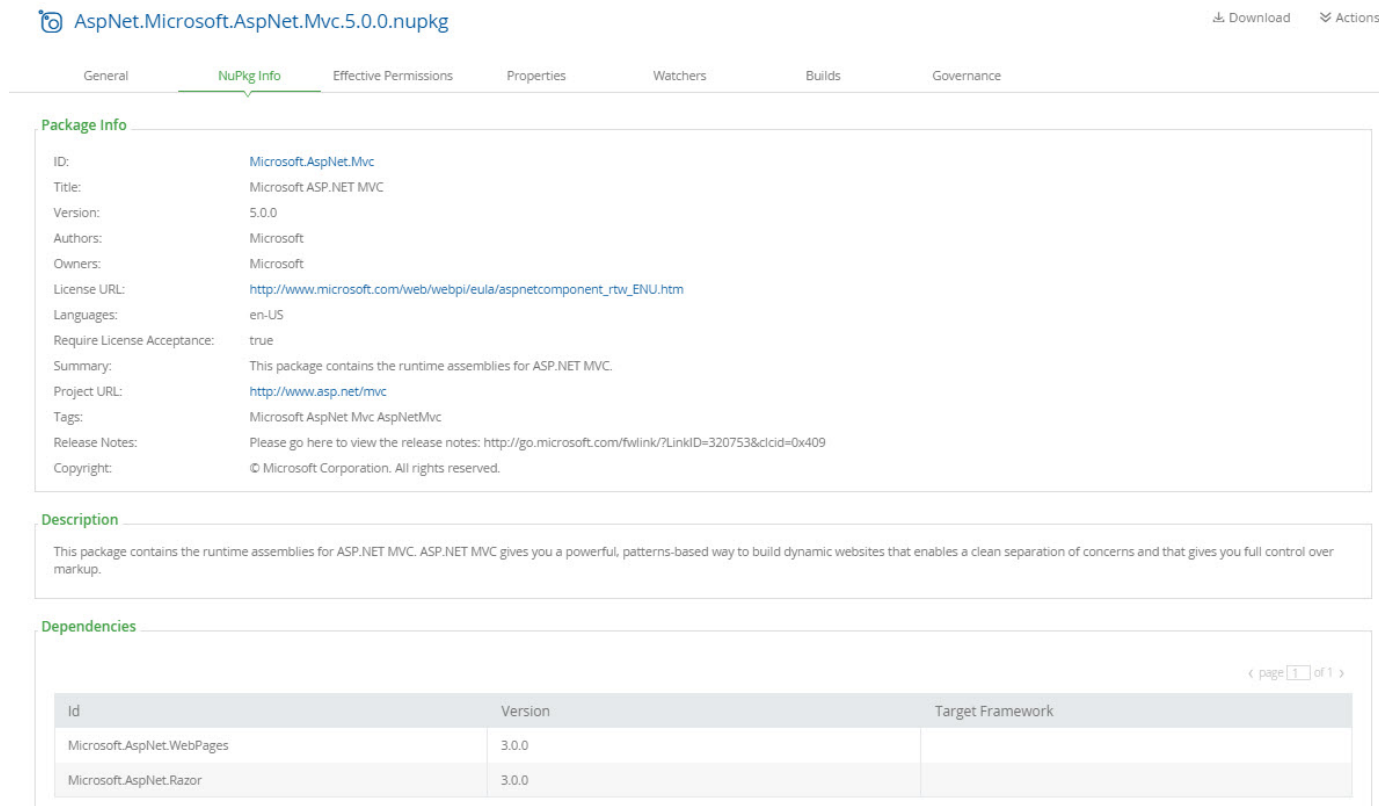
Max Unique Snapshots [?](#)

Force Authentication [?](#)

When set, Artifactory will first request authentication parameters from the NuGet client before trying to access this repository.

Viewing Individual NuGet Package Information

You can view all the metadata annotating a NuGet package by choosing the NuPkg file in Artifactory's tree browser and selecting the NuPkg Info tab:



The screenshot shows the Artifactory interface for the package `AspNet.Microsoft.AspNet.Mvc.5.0.0.nupkg`. The `NuPkg Info` tab is selected, displaying the following metadata:

- ID: `Microsoft.AspNet.Mvc`
- Title: `Microsoft ASP.NET MVC`
- Version: `5.0.0`
- Authors: `Microsoft`
- Owners: `Microsoft`
- License URL: http://www.microsoft.com/web/webpi/eula/aspnetcomponent_rtw_ENU.htm
- Languages: `en-US`
- Require License Acceptance: `true`
- Summary: `This package contains the runtime assemblies for ASP.NET MVC.`
- Project URL: <http://www.asp.net/mvc>
- Tags: `Microsoft.AspNet.Mvc.AspNet.Mvc`
- Release Notes: [Please go here to view the release notes: http://go.microsoft.com/fwlink/?LinkId=320753&clcid=0x409](http://go.microsoft.com/fwlink/?LinkId=320753&clcid=0x409)
- Copyright: `© Microsoft Corporation. All rights reserved.`

The `Description` section states: "This package contains the runtime assemblies for ASP.NET MVC. ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and that gives you full control over markup."

The `Dependencies` section shows a table with the following data:

Id	Version	Target Framework
<code>Microsoft.AspNet.WebPages</code>	<code>3.0.0</code>	
<code>Microsoft.AspNet.Razor</code>	<code>3.0.0</code>	

Watch the Screencast

Microsoft Symbol Server

Overview

Microsoft Symbol Server is a Windows technology used to obtain debugging information (symbols) needed in order to debug an application with various Microsoft tools. Symbol files (which are .pdb files) provide a footprint of the functions that are contained in executable files and dynamic-link libraries (DLLs), and can present a roadmap of the function calls that lead to the point of failure.

A Symbol Server stores the .PDB files and binaries for all your public builds. These are used to enable you to debug any crash or problem that is reported for one of your stored builds. Both Visual Studio and WinDBG know how to access Symbol Servers, and if the binary you are debugging is from a public build, the debugger will get the matching PDB file automatically.

The TFS 2010 build server includes built-in build tasks to index source files and copy symbol files to your Symbol Server automatically as part of your build.

Page Contents

- Overview
- Using Artifactory as Your Symbol Server
 - Configuring Your Debugger
 - Configuring Your Build
 - Configure Repositories in Artifactory
 - Install the Artifactory Symbol Server Plugin
 - Configure IIS
 - Configuring Visual Studio

Using Artifactory as Your Symbol Server

Configuring your system to use Artifactory as your Symbol Server requires the following main steps:

1. Configure your debugger
2. Configure your build
3. Configure repositories in Artifactory
4. Install the Artifactory Symbol Server Plugin
5. Configure IIS
6. Configure Visual Studio

Configuring Your Debugger

To enable you to step into and debug your source files, your debugger needs the corresponding `.pdb` files and searches for them in the following order until they are found:

1. The directory from which your binary was loaded
2. The hard-coded build directory specified in the **Debug Directories** entry of your portable executable (PE) file
3. Your Symbol Server cache directory (assuming you have a Symbol Server set up)
4. The Symbol Server itself

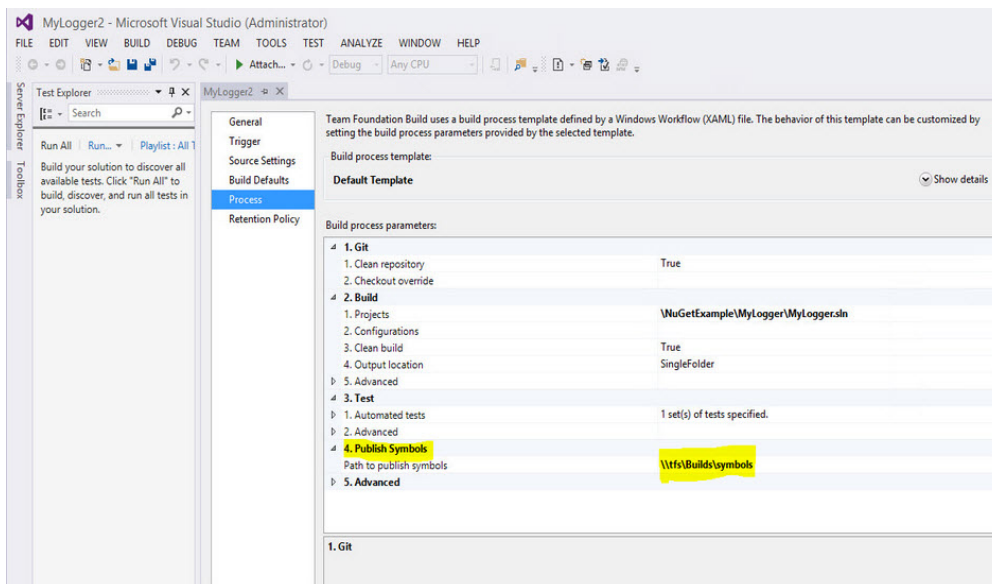
So to fully support this search order, you need to specify the Symbol Server URL in your debugger.

Under **Tools | Options | Debugging | Symbols** enter <http://msdl.microsoft.com/download/symbols>

Configuring Your Build

You need to configure your build machine to publish your `.pdb` files into a known directory which is later used in your IIS configuration.

Assuming you are using TFS, and want to publish your `.pdf` files into a directory called `Builds/symbols`, your build definition would look something like the below:



Configure Repositories in Artifactory

Create the following repositories:

Repository name	Description/Instructions
microsoft-symbols-IIS	A Remote repository. Set the repository URL to point to the virtual directory configured in your IIS below. (For the example on this page we will use <code>http://localhost/symbols</code>)
microsoft-symbols	A Remote repository. Set the repository URL to point to the Microsoft Symbol Server URL: <code>http://msdl.microsoft.com/download/symbols</code> .
symbols	A virtual repository. Configure this repository to aggregates the other two repositories, resolving from microsoft-symbols-IIS first. Once configured, this repository will aggregate all the NuGet packages with symbol (.pdb) files. <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"><p>Make this a NuGet repository Be sure to specify NuGet as the Package Type for this repository</p></div>

Install the Artifactory Symbol Server Plugin

The Artifactory Symbol Server Plugin listens for requests for symbol files and then redirects them to the Microsoft Symbol Server.

Download the [Artifactory Symbol Server Plugin from GitHub](#) and install it in your `$ARTIFACTORY-HOME\etc\plugin` directory.

Check your system properties

To ensure that the plugin is loaded, check that your `artifactory.plugin.scripts.refreshIntervalSecs` system property is not 0.

If you do modify this system property, you need to restart Artifactory for this modification to take effect.

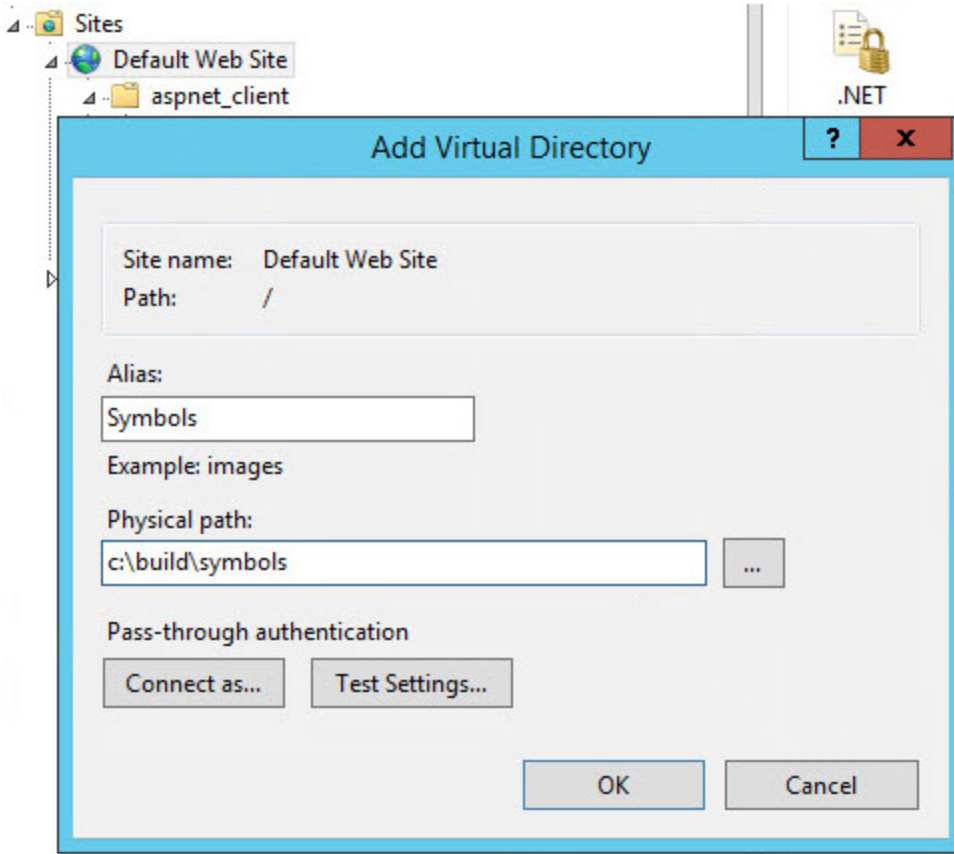
Configure IIS

To configure your Internet Information Services (IIS) machine, you need to

- Add a virtual directory on which your `.pdb` files reside
- Define a MIME type to be associated with `.pdb` files.
- Enable directory browsing.

To add a virtual directory on your IIS, execute the following steps:

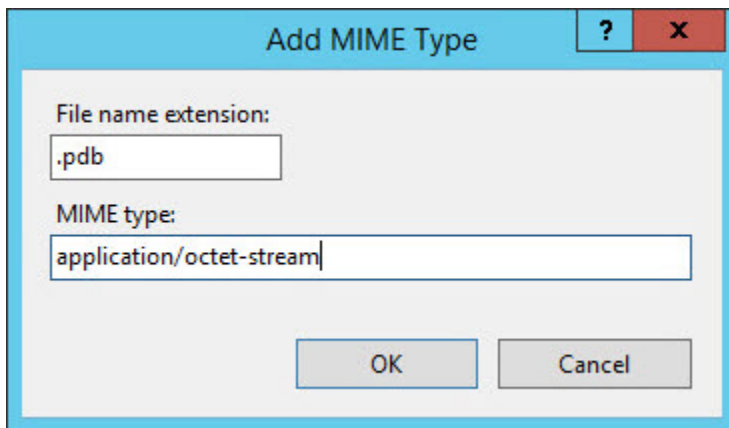
- Under **Control Panel | System and Security | Administrative Tools | Internet Information Service(IIS) Manager**, right click **Default Website** and select **Add Virtual Directory**.
- Set **Physical path** to `C:\build\symbols`



To define a MIME type so that your IIS associates the `.pdb` file extension with Symbol files, execute the following steps:

- Right click the MIME Type symbol and select **Open Feature**.
- Click **Add...** on the right side of the window fill in the fields as follows:

Field	Value
File name extension	.pdb
MIME type	application/octet-stream



Configuring Visual Studio

Before you configure Visual Studio, you need to remove the symbol cache located under `C:\Users\Administrator\AppData\Local\Temp\1\SymbolCache`

Once you have removed the symbol cache, you need to change the location of the symbol (*.pdb*) file. Under **Tools | Options | Debugging | Symbols** add a new symbol server pointing to the **symbols** virtual directory you defined in [Artifactory](#) above.

This should be `http://tfs::8081/artifactory/symbols`

Note that there is no way to set the path directly to Artifactory since the symbol server cannot take a URL as a path.

Opkg Repositories

Overview

From version 4.4, Artifactory supports Opkg repositories. As a fully-fledged Opkg repository, Artifactory generates index files that are fully compliant with the Opkg client.

Artifactory support for Opkg provides:

1. The ability to provision ipk packages from Artifactory to an Opkg client from local and remote repositories.
2. Calculation of Metadata for ipk packages hosted in [Local Repositories](#).
3. Access to remote Opkg resources (such as *downloads.openwrt.com*) through [Remote Repositories](#) which provide the usual proxy and caching functionality.
4. Providing GPG signatures that can be used by Opkg clients to verify packages.
5. Complete management of GPG signatures using the Artifactory UI and the REST API.

Page Contents

- [Overview](#)
- [Configuration](#)
 - [Local Repositories](#)
 - [Deploying a package using the UI](#)
 - [Remote Repositories](#)
- [Configuring the Opkg Client to Work with Artifactory](#)
- [Signing Opkg Package Indexes](#)
- [Authenticated Access to Servers](#)
- [REST API Support](#)

Configuration

You can only deploy Opkg packages to a local repository that has been created with the Opkg **Package Type**.

You can download packages from a local or a remote Opkg repository.

Local Repositories

To create a new local repository that supports Opkg, under the **Basic** settings, set the **Package Type** to be **Opkg**.

Artifactory supports the common Opkg index scheme which indexes each feed location according to all ipk packages in it.

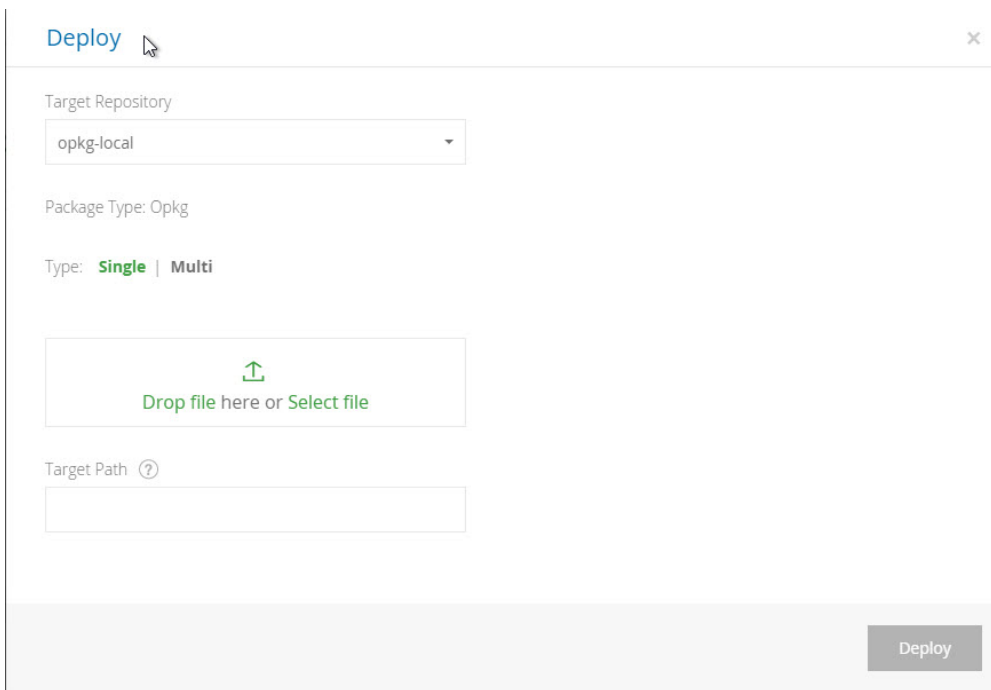


The screenshot shows the 'New Local Repository' dialog box with the 'Basic' tab active. The 'Package Type' dropdown is set to 'Opkg', which is highlighted with a red box. Below it is a text input field for 'Repository Key'.

Deploying a package using the UI

To deploy a Opkg package to Artifactory, in the Artifactory Repository Browser, click **Deploy**.

Select your Opkg repository as the **Target Repository**, and upload the file you want to deploy.



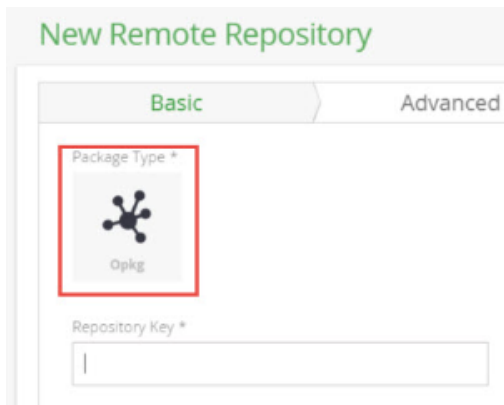
The screenshot shows the 'Deploy' dialog box. The 'Target Repository' dropdown is set to 'opkg-local'. The 'Package Type' is 'Opkg'. The 'Type' is set to 'Single'. There is a file upload area with a green arrow icon and the text 'Drop file here or Select file'. Below it is a 'Target Path' input field. A 'Deploy' button is visible at the bottom right.

After you deploy the artifact, you need to wait about one minute for Artifactory to recalculate the repository index and display your upload in the Repository Browser.

Remote Repositories

You can download ipk packages from Local Opkg Repositories as described above, or from Remote Repositories specified as supporting Opkg packages.

To specify that a Remote Repository supports Opkg packages, you need to set its **Package Type** to **Opkg** when it is created.



You can either point the remote to a specific feed (location of a Packages file), i.e. http://downloads.openwrt.org/chaos_calmer/15.05/adm5120/rb1xx/packages/luci

Or you can specify some base level and point your client to the relevant feeds in it i.e. url is http://downloads.openwrt.org/chaos_calmer/15.05/ and your opkg.conf file has the entry `src adm5120/rb1xx/packages/luci`

Note that the index files for remote Opkg repositories are stored and renewed according to the [Retrieval Cache Period](#) setting.

Configuring the Opkg Client to Work with Artifactory

As there is no "release" of the Opkg client, to support gpg signature verification and basic HTTP authentication that are provided by Artifactory it has to be compiled with the following options: `--enable-gpg --enable-curl`

For example, to compile Opkg on Ubuntu to support these you can use:

Compiling Opkg

```
# Download opkg release (latest when this was written was 0.3.1):
wget http://downloads.yoctoproject.org/releases/opkg/opkg-0.3.1.tar.gz
tar -zxvf opkg-0.3.1.tar.gz
# Install compilation dependencies:
apt-get update && apt-get install -y gcc libtool autoconf pkg-config
libarchive13 libarchive-dev libcurl3 libcurl4-gnutls-dev libssl-dev
libpgm11-dev
# Compile Opkg(compile with curl to support basic auth, and with gpg
support for signature verification):
# Note: if there's no configure script in the release you downloaded you
need to call ./autogen.sh first
./configure --with-static-libopkg --disable-shared --enable-gpg
--enable-curl --prefix=/usr && make && sudo make install
```

Each Opkg feed corresponds to a path in Artifactory where you have chosen to upload ipk packages to. This is where the Packages index is written.

For example, you can add each such feed to your `opkg.conf` (default location is `/etc/opkg/opkg.conf`) file with entries like:

Opkg feed locations

```
src artifactory-armv7a
http://prod.mycompany:8080/artifactory/opkg-local/path/to/my/ipks/armv7a
src artifactory-i386
http://prod.mycompany:8080/artifactory/opkg-local/path/to/my/ipks/i386
```

Signing Opkg Package Indexes

Artifactory uses your GPG public and private keys to sign and verify Opkg package indexes.

To learn how to generate a GPG key pair and upload it to Artifactory, please refer to [GPG Signing](#).

Once you have GPG key pair, to have Opkg verify signatures created with the private key you uploaded to Artifactory, you need to import the corresponding public key into Opkg's keychain (requires *gnupg*).

Importing gpg keys to Opkg's keychain

```
# Commands taken from opkg-utils package:
mkdir /etc/opkg
gpg --no-options --no-default-keyring --keyring /etc/opkg/trusted.gpg
--secret-keyring /etc/opkg/secring.gpg --trustdb-name /etc/opkg/trustdb.gpg
--batch --import key.pub
```

After the key is imported you need to add the `check_signature` option in your `opkg.conf` file by adding the following entry:

Opkg signature verification

```
option check_signature true
```

Authenticated Access to Servers

If you need to access a secured Artifactory server that requires a username and password, you can specify these in your `opkg.conf` file by adding the `'http_auth'` option:

Accessing Artifactory with credentials

```
option http_auth user:password
```

Encrypting your password

You can use your encrypted password as described in [Using Your Secure Password](#).

REST API Support

The Artifactory REST API provides extensive support for signing keys and recalculating the repository index as follows:

- Set the public key
- Get the public key
- Set the private key
- Set the pass phrase
- Recalculate the index

P2 Repositories

Overview

From version 2.4, Artifactory provides advanced support for proxying and caching of P2 repositories and aggregating P2 metadata using an Artifactory virtual repository which serves as a single point of distribution (single URL) for Eclipse, [Tycho](#) and any other P2 clients.

This virtual repository aggregates P2 metadata and P2 artifacts from underlying repositories in Artifactory (both local and remote) providing you with full visibility of the P2 artifact sources and allowing powerful management of caching and security for P2 content.

For more information on defining virtual repositories please refer to [Virtual Repositories](#).

For P2 support we recommend using Eclipse Helios (version 3.6) and above.
Older versions of Eclipse may not work correctly with Artifactory P2 repositories.

Page Contents

- Overview
- Configuration
 - Defining a Virtual Repository
 - Selecting Local Repositories
 - Selecting Remote Repositories
 - Creating the Repositories
 - Eclipse
- Integration with Tycho Plugins
- Multiple Remote Repositories with the Same Base URL
- Configuring Google Plugins Repository

Configuration

To use P2 repositories, follow the steps below:

- Define a virtual repository in Artifactory
- Select local repositories to add to your virtual repository
- Select remote repositories to add to your virtual repository
- Create the selected local and remote repositories in your virtual repository
- Configure Eclipse to work with your virtual repository

Defining a Virtual Repository

- Create a new virtual repository and set **P2** as the **Package Type**

New Virtual Repository

Basic

Advanced

Package Type *



Repository Key *

p2-virtual

If developers in your organization use different versions of Eclipse (e.g. Helios and Juno), we recommend that you define a different P2 virtual repository for each Eclipse version in use.

Selecting Local Repositories

Adding a local repository to your virtual P2 repository does not require any special configuration:

- Simply select the desired local repository from the **Local Repository** field. Usually, this will be either a Maven or a Generic repository.
- In the **Path Prefix** field, specify the path to the P2 metadata files (`content.jar`, `artifacts.jar`, `compositeContent.xml` etc.). If left empty, the default is to assume that the P2 metadata files are directly in the repository root directory.
- Click the "Add" button.

Local P2 Repositories

Local Repository

p2-local

Path Suffix

path/to/metadata

Add

If you have a Tycho repository deployed to a local repository as a single archive, specify the archive's root path. For example: `eclipse-repository.zip/!`

Local P2 Repositories

Local Repository

p2-tycho-local

Path Suffix

eclipse-repository.zip/

Add

Selecting Remote Repositories

To add a remote P2 repository to Artifactory, enter the URL to the corresponding P2 metadata files (`content.jar`, `artifacts.jar`, `compositeContent.xml`, etc.) and click the "Add" button

Two common examples are:

1. The main Juno repository: <http://download.eclipse.org/releases/juno>
2. The Google plugins repository for Indigo (GWT, GAE, etc.): <http://dl.google.com/eclipse/plugin/3.7>

Remote P2 Repositories

P2 Repository URL

<http://download.eclipse.org/releases/juno>

Add

Artifactory analyzes the added URL and identifies which remote repositories should be created in Artifactory based on the remote P2 metadata (since remote P2 repositories may aggregate information from different hosts).

When P2 metadata files reside inside an archived file, simply add "!" to the end of the URL.

For example: <http://eclipse.org/equinox-sdk.zip/>

Creating the Repositories

Once you have selected the local and remote repositories to include in your virtual repository, Artifactory will indicate what action will be taken once you select the "Save & Finish" button.

The possible actions are as follows:

Create*	Creates a new, P2 enabled, remote repository with the given key (you may still edit the remote repository key).
Modify*	Enables P2 support in an existing remote repository.
Include	Adds the repository to the list of repositories aggregated by this virtual repository.
Included	No action will be taken. This repository is already included in the virtual repository.

*For remote repositories only

Filter by Repository

< page 1 of 1 >

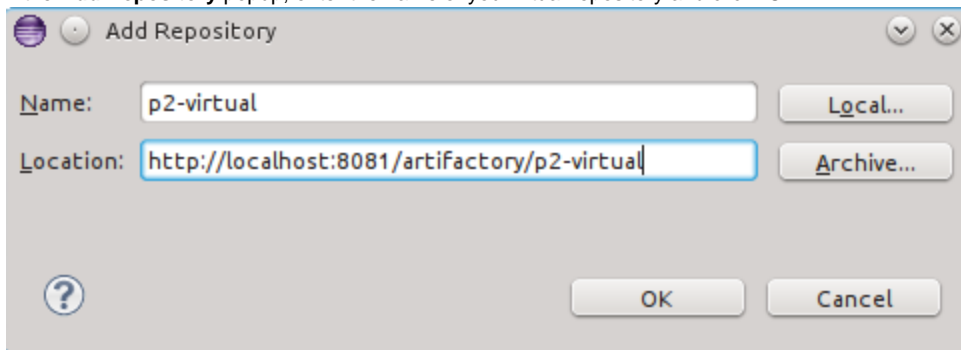
Action	Repository	URL
include	download.eclipse.org	http://download.eclipse.org
include	p2-local	local://p2-local/path/to/metadata
include	p2-tycho-local	local://p2-tycho-local/eclipse-repositor...

Eclipse

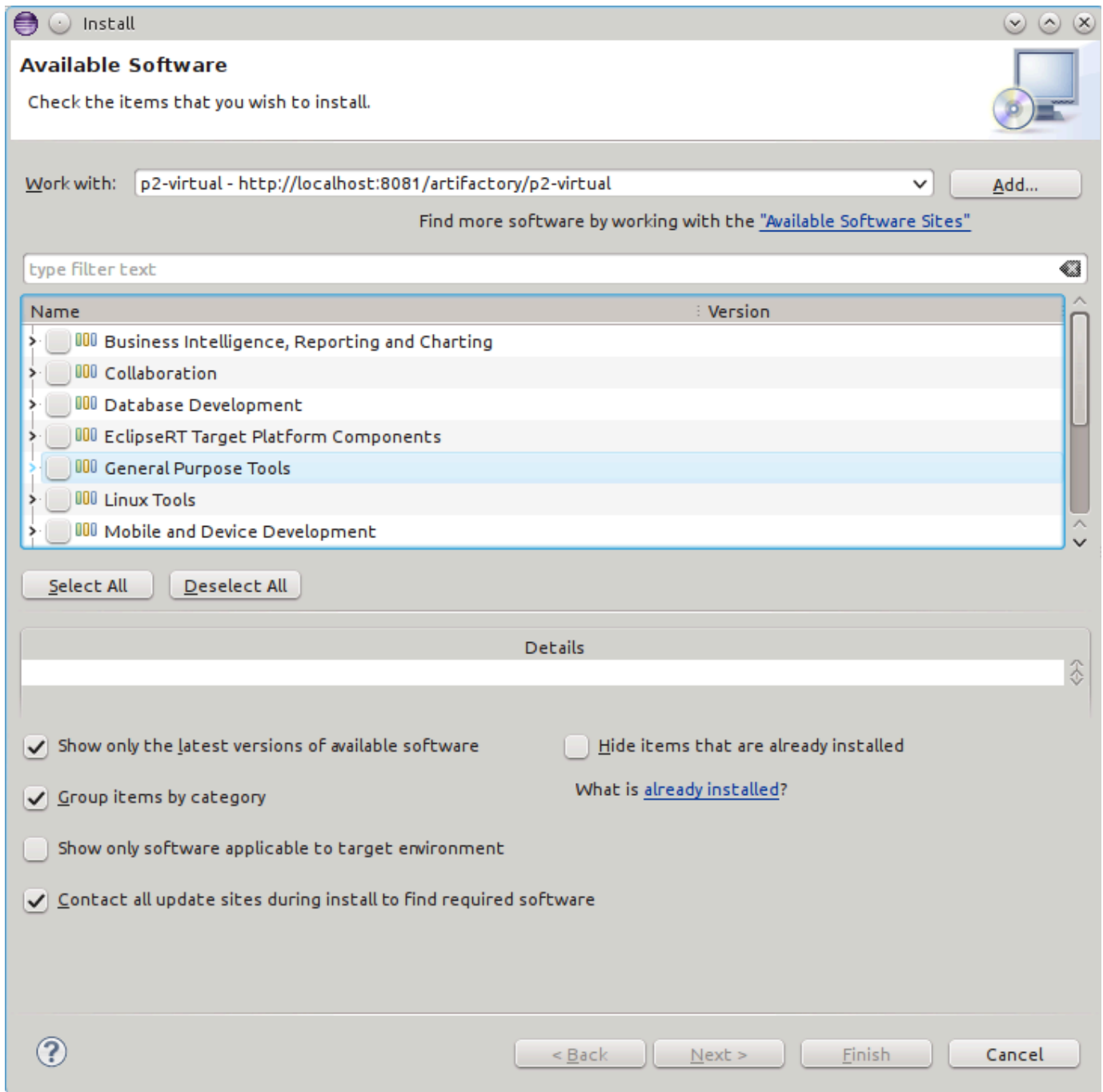
You are now ready to configure eclipse to work with the virtual repository you have created above.

In the Eclipse menu, select **Help | Install new Software** and then click **Add**.

In the **Add Repository** popup, enter the name of you virtual repository and click "OK":



Eclipse will then query Artifactory for available packages and update the screen to display them as below:



Integration with Tycho Plugins

Artifactory fully supports hosting of Tycho plugins as well as resolving Tycho build dependencies.

To resolve all build dependencies through Artifactory, simply change the repository URL tag of your build pom.xml file and point it to a dedicated virtual repository inside Artifactory

For example:

```

<repository>
  <id>eclipse-indigo</id>
  <layout>p2</layout>
  <url>http://localhost:8081/artifactory/p2-virtual</url>
</repository>

```

The P2 virtual repository should contain URLs to all local repositories with an optional sub-path in them where Tycho build artifacts reside.

Multiple Remote Repositories with the Same Base URL

When using P2-enabled repositories with multiple remote repositories that have the same base URL (e.g <http://download.eclipse.org>), you need to ensure that only 1 remote repository is created within your virtual repository (for each base URL). When creating your virtual repository, Artifactory takes care of this for you, but if you are creating the remote repositories manually, you must ensure to create only a single remote repository, and point the sub-paths accordingly in the P2 virtual repository definition.

In the example below, <http://download.eclipse.org/releases/helios> and <http://download.eclipse.org/releases/juno> were both added to the same virtual repository...repository.

Filter by Repository < page 1 of 1 >

Action	Repository	URL
included	download.eclipse.org	http://download.eclipse.org/releases/helios
included	download.eclipse.org	http://download.eclipse.org/releases/juno

...but in fact, the virtual repository only really includes one remote repository

Virtual Repositories

Filter by Repository key or Type

Repository key	Type	Included Repositories	Selected Repositories
p2-virtual-multiremote	P2	1	download.eclipse.org
plugins-release	Maven	3	plugins-release-local;ext-release-local;remote-repos
plugins-snapshot	Maven	3	plugins-snapshot-local;ext-snapshot-local;remote-repos
remote-repos	Maven	3	java.net.m1;repo1;gradle-libs

Configuring Google Plugins Repository

The Google Plugins repository (<http://dl.google.com/eclipse/plugin/3.7>) is aggregated across 3 different URLs. Therefore you need to configure Artifactory to create all of them in order to resolve P2 artifacts correctly:

Action	Repository	URL
create	dl-ssl.google.com	http://dl-ssl.google.com
create	dl.google.com	http://dl.google.com
create	dl.google.com-1	https://dl.google.com-1

PHP Composer Repositories

Overview

Artifactory supports [PHP Composer](#) repositories on top its existing support for advanced artifact management.

Artifactory support for Composer provides:

1. The ability to provision Composer packages from Artifactory to the Composer command line tool from all repository types.
2. Calculation of metadata for Composer packages hosted in Artifactory local repositories.
3. Access to remote Composer metadata repositories (Packagist and Artifactory Composer repositories) and package repositories (such as Github, Bitbucket etc..) through [remote repositories](#) which provide proxy and caching functionality.
4. Assign access privileges according to projects or development teams.

Configuration

Local Repositories

To enable calculation of Composer package metadata, set **PHP Composer** to be the **Package Type** when you create your local Composer repository.



Deploying Composer Packages

The Composer client does not provide a way to deploy packages and relies on a source control repository to host the Composer package code. To deploy a Composer package into Artifactory, you need to use Artifactory's [REST API](#) or the [Web UI](#).

A Composer package is a simple archive, usually zip or a tar.gz file, which contains your project code as well as a `composer.json` file describing the package.

Page Contents

- Overview
- Configuration
 - Local Repositories
 - Deploying Composer Packages
 - Remote Repositories
- Using the Composer command line
 - Replacing the Default Repository
 - Authentication
- Cleaning Up the Local Composer Cache
- Viewing Individual Composer Package Information

Version

For Artifactory to index packages you upload, each package must have its version specified. There are three ways to specify the

package version:

- Include the `version` attribute in the package `composer.json` file
- Set a `composer.version` property when deploying a package via REST (or on an existing package)
- Use the **version** field when deploying via the UI

Remote Repositories

The [public Composer repository](#) does not contain any actual binary packages; it contains the package indexes that point to the corresponding source control repository where the package code is hosted.

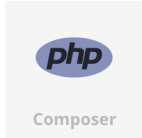
Since the majority of public Composer packages are hosted on GitHub, we recommend creating a Composer [remote repository](#) to serve as a caching proxy for [github.com](#), specifying [packagist.org](#) as the location of the public package index files. A Composer remote repository in Artifactory can proxy [packagist.org](#) and other Artifactory Composer repositories for index files, and version control systems such as GitHub or BitBucket, or local Composer repositories in other Artifactory instances for binaries.

Composer artifacts (such as zip, tar.gz files) requested from a remote repository are cached [on demand](#). You can remove the downloaded artifacts from the remote repository cache, however you can not manually deploy artifacts to a remote repository.

To define a remote repository to proxy [github.com](#) as well as the public Composer Packagist repository follow the steps below:

1. Create a new remote repository and set **PHP Composer** to be its **Package Type**
2. Set the **Repository Key**, and enter the repository URL (e.g. <https://github.com/>) in the **URL** field as displayed below
3. In the **Composer Settings** section, select **GitHub** as the **Git Provider**, and leave the default **Registry URL** (e.g. <https://packagist.org/>).
4. Finally, click "Save & Finish"

Package Type *



Repository Key *

composer-github

URL *

https://github.com/

Test

General

Repository Layout

composer-default

Composer Settings

Git Provider

GitHub

Remote Layout Mapping

Select Remote Layout

Registry URL

https://packagist.org

URL vs. Registry URL

To avoid confusion, note that:

URL is the URL of your Git provider where the actual package binaries are hosted.

Registry URL is the URL where the package index files holding the metadata are hosted.

To proxy a public Composer registry, set the Registry URL field to the location of the index files as displayed above. To proxy a Composer repository in another Artifactory instance, set both the **URL** field and the **Registry URL** field to the remote Artifactory repository's API URL. For example: <https://frog-art.com/artifactory/api/composer/composer-local>

Using the Composer command line

Once the Composer client is installed, you can access Composer repositories in Artifactory through its command line interface.

Composer repositories must be prefixed with `api/composer` in the path

When accessing a Composer repository through Artifactory, the repository URL must be prefixed with `api/composer` in the path. This applies to all Composer commands including `composer install`.

For example, if you are using Artifactory standalone or as a local service, you would access your Composer repositories using the following URL:

```
http://localhost:8081/artifactory/api/composer/<repository key>
```

Or, if you are using Artifactory SaaS, the URL would be:

```
https://<server name>.jfrog.io/<server name>/api/composer/<repository key>
```

Once you have created a Composer repository, you can select it in the Tree Browser and click **Set Me Up** to get code snippets you can use to set your Composer repository URL in your `config.json` file.

Set Me Up ✕

Tool

php PHP Composer

Insert Credentials

Repository

composer-local

General

In order to configure your Composer client to work with Artifactory, you need to edit its `config.json` file (which can usually be found under `<user-home-dir>/composer/config.json`) and add a repository reference to your Artifactory Composer repository. For example:

```
1 {
2   "repositories": [
3     {"type": "composer", "url": "http://localhost:8081/artifactory/api/composer/composer-local"},
4     {"packagist": false}
5   ]
6 }
```

Composer config.json file

Windows: `%userprofile%\composer\config.json`

Linux: `~/.composer/config.json`

Replacing the Default Repository

You can change the default repository specified for the Composer command line in the `config.json` file as follows:

```
{
  "repositories": [
    {
      "type": "composer",
      "url":
      "https://localhost:8081/artifactory/api/composer/composer-local"},
    {
      "packagist": false
    }
  ]
}
```

Working with a secure URL (HTTPS) is considered a best practice, but you may also work with an insecure URL (HTTP) by setting the *secure-http* configuration to *false*:

```
{
  "config": {
    "secure-http" : false
  },
  "repositories": [
    ...
  ]
}
```

Authentication

In order to authenticate the Composer client against your Artifactory server, you can configure Composer to use basic authentication in your *auth.json* file as follows:

```
{
  "http-basic": {
    "localhost": {
      "username": "mikep",
      "password": "APBJ7XgkrigBzb2XKTuwgnRq5vc"
    }
  }
}
```

Composer auth.json file

Windows: %userprofile%\composer\auth.json

Linux: ~/.composer/auth.json

Once the Composer command line tool is configured, every `composer install` command will fetch packages from the Composer repository specified above.

Cleaning Up the Local Composer Cache

The Composer client saves caches of packages that were downloaded, as well as metadata responses. We recommend removing the Composer caches (both packages and metadata responses) before using Artifactory for the first time, this is to ensure that your caches only contain elements that are due to requests from Artifactory and not directly from Packagist. To clear your Composer cache, run the following command:

Clean the Composer cache

```
composer clear-cache
```

composer.lock file

In your project directory already has a *composer.lock* file that contains different 'dist' URLs (download URLs) than Artifactory, you need to remove it, otherwise, when running the *composer install* command, the composer client will resolve the dependencies using the *composer.lock* file URLs

Viewing Individual Composer Package Information

Artifactory lets you view selected metadata of a Composer package directly from the UI.

In the **Artifacts** tab, select **Tree Browser** and drill down to select the package archive file you want to inspect. The metadata is displayed in the **Composer Info** tab.

The screenshot shows the Artifactory UI with the 'Composer Info' tab selected. The interface includes a navigation bar with tabs: 'General', 'Composer Info', 'Effective Permissi...', 'Properties', and 'Watchers'. The 'Composer Info' tab is active, displaying the following information:

- Package Info**
 - Name: jfrog/printer
 - Version: 1.0.0
 - Type: library
 - Keywords: printer, frog
 - Licenses: MIT
 - Authors: Shay Bagants
- Description**
 - My new printer package
- Require**
 - 1 Record

Name	Version
jfrog/log	1.0.0

Puppet Repositories

Overview

Artifactory provides full support for managing Puppet modules, ensuring optimal and reliable access to **Puppet Forge**. By aggregating multiple Puppet repositories under a single virtual repository Artifactory enables upload and download access to all your Puppet modules through a single URL.

As a fully-fledged Puppet repository, on top of its capabilities for [advanced artifact management](#), Artifactory's support for [Puppet](#) provides:

1. The ability to provision Puppet modules from Artifactory to the Puppet command line tool for all repository types.

Page Contents

- [Overview](#)

2. Calculation of Metadata for Puppet modules hosted in Artifactory's local repositories.
3. Access to remote Puppet repositories, such as <https://forgeapi.puppetlabs.com/>, using [Remote Repositories](#) which provides proxy and caching functionalities.
4. Access to multiple Puppet repositories from a single URL by aggregating them under a [Virtual Repository](#). This overcomes the limitation of the Puppet client which can only access a single registry at a time.
5. Support for [flexible puppet repository layouts](#) that allow you to organize your Puppet modules, and assign access privileges according to projects or development teams.

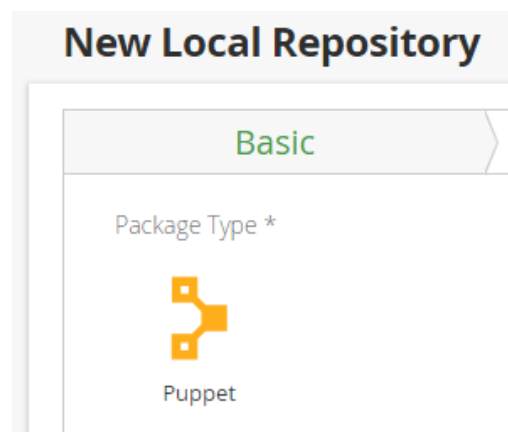
Puppet version support

Puppet does not support a context path up to version 4.9.1, we recommend using Artifactory with Puppet version **4.9.2** and above. Please see below if you are using Puppet 4.9.1 and below.

Configuration

Local Puppet Repository

To enable calculation of Puppet module metadata in local repositories, set the **Package Type** to **Puppet** when you create the repository:



- Configuration
 - Local Puppet Repository
 - Repository Layout
 - Remote Puppet Repository
 - Virtual Puppet Repository
- Using the Puppet Command Line
- Using librarian-puppet
- Using r10k
- Puppet Publish (Deploying Modules)
 - Setting Your Credentials
 - Deploying Your Modules
- Working with Artifactory without Anonymous Access
- Puppet Search
- Cleaning Up the Local Puppet Cache
- Viewing Individual Puppet Module Information
- Using Puppet 4.9.1 and Below

- REST API
 - Get Puppet Modules
 - Get Puppet Module
 - Get Puppet Releases
 - Get Puppet Release

Repository Layout

Artifactory allows you to define any layout for your Puppet repositories. To upload a module according to your custom layout, you need to package your Puppet files using `puppet module build`.

This creates a **.tar.gz** file for your module which you can then upload to any path within your local Puppet repository.

Remote Puppet Repository

A [Remote Repository](https://forgeapi.puppetlabs.com/) defined in Artifactory serves as a caching proxy for a repository managed at a remote URL such as <https://forgeapi.puppetlabs.com/>.

Artifacts (such as tar.gz files) requested from a remote repository are cached on demand. You can remove downloaded artifacts from the remote repository cache, however, you can not manually deploy artifacts to a remote Puppet repository.


To define a remote repository to proxy a remote Puppet resource follow the steps below:

1. In the **Admin** module, under **Repositories | Remote**, click "New".
2. In the New Repository dialog, set the **Package Type** to **Puppet**, set the **Repository Key** value, and specify the URL to the remote repository in the **URL** field as displayed below.
3. Click "Save & Finish".

New Remote Repository

Basic | Advanced | Replications

Package Type *

 Puppet

Repository Key *

URL *

Virtual Puppet Repository

A **virtual** repository, in Artifactory, aggregates modules from both **local** and **remote** repositories.

This allows you to access both locally hosted Puppet modules and those from remote proxied Puppet repositories from a single URL defined for the virtual repository.

To define a virtual Puppet repository, create a [virtual repository](#), set the **Package Type** to **Puppet**, and select the underlying local and remote Puppet repositories to include in the **Basic** settings tab.

Click "Save & Finish" to create the repository.

Repositories

Available Repositories

<<<>>>

Selected Repositories

- 🔍 puppet-local✕
- 🔍 puppet-remote✕

Included Repositories

When configuring the order of resolution note that Artifactory will always resolve first from local repositories, then cache and only then will try to request artifacts from remote repository.

puppet-local

puppet-remote

Default Deployment Repository

puppet-local▼

Using the Puppet Command Line

When accessing a Puppet repository through Artifactory, the repository URL path must be prefixed with **api/puppet**.

This applies to all Puppet commands including `puppet module install` and `puppet module search`.

For example, if you are using Artifactory standalone or as a local service, you would access your Puppet repositories using the following URL:

```
http://localhost:8081/artifactory/api/puppet/<REPO_KEY>
```

Or, if you are using Artifactory SaaS the URL would be:

```
https://<server name>.jfrog.io/<server name>/api/puppet/<REPO_KEY>
```

To use the Puppet command line you need to make sure Puppet is installed on your client.

Once you have created your Puppet repository, you can select it in the Tree Browser and click the **Set Me Up** button to get useful code snippets. These allow you to change your Puppet repository URL in the **puppet.conf** file, and resolve modules using the Puppet command line tool.

Set Me Up ✕

Tool

🔗 Puppet

Repository

puppet

🔒 Type password to insert your credentials to the code snippets

➔

General

In order for your Puppet client to work with Artifactory you will need to add following in your puppet.conf file:

```

1 [main]
2 module_repository=http://<USERNAME>:<PASSWORD>@localhost:8080/artifactory/api/puppet/puppet

```

Deploy

To deploy a Puppet module into an Artifactory repository you need to use Artifactory's REST API or the Web UI. For example, to deploy a Puppet module into this repository using the REST API, use the following command:

```

1 curl -u<USERNAME>:<PASSWORD> -XPUT http://localhost:8080/artifactory/puppet/<TARGET_FILE_PATH> -T <PATH_TO_FILE>

```

Resolve

To install a module by specifying Artifactory repository use the following puppet command:

```

1 puppet module install --module_repository=http://<USERNAME>:<PASSWORD>@localhost:8080/artifactory/api/puppet/puppet <MODULE_NAME>

```

Replacing the default repository

To replace the default repository with a URL pointing to a Puppet repository in Artifactory, add following in your **puppet.conf** file:
 Note: This example uses a repository with the key **puppet**

```

[main]
module_repository=http://localhost:8080/artifactory/api/puppet/puppet

```

Tip: We recommend referencing a Virtual Repository URL as a repository. This gives you the flexibility to reconfigure and aggregate other external sources and local repositories of Puppet modules you deploy.

Note that if you do this, you can also use the `--module_repository` parameter to specify the local repository from which you want to resolve your module when using the Puppet module install command.

Once the Puppet command line tool is configured, every `puppet module install` command will fetch packages from the Puppet repository specified above. For example:


```
$ puppet module install
--module_repository=http://localhost:8080/artifactory/api/puppet/puppet
puppetlabs-mysql
Notice: Preparing to install into
/Users/jainishs/.puppetlabs/etc/code/modules ...
Notice: Downloading from
http://localhost:8080/artifactory/api/puppet/puppet ...
Notice: Installing -- do not interrupt ...
/Users/jainishs/.puppetlabs/etc/code/modules
puppetlabs-mysql (v3.10.0)
```

Using librarian-puppet

librarian-puppet is a bundler for your Puppet infrastructure. From version 5.4.5, you can use librarian-puppet with Artifactory as a Puppet repository to manage the Puppet modules your infrastructure depends on.

To configure librarian-puppet to fetch modules from Artifactory, add the following to your `Puppetfile`:

```
forge
"http://<ARTIFACTORY_HOST_NAME>:<ARTIFACTORY_PORT>/artifactory/api/puppet/<REPO_KEY>"
```

For example, a Puppetfile that uses librarian-puppet could look something like this:

```
forge "http://localhost:8080/artifactory/api/puppet/puppet-local"

mod 'puppetlabs-mysql', '3.7.0'
mod 'puppetlabs-apache', '1.5.0'
```

To fetch and install the Puppet modules from Artifactory, run the following command:

```
librarian-puppet install --no-use-v1-api
```

Using r10k

r10k is a Puppet environment and module deployment tool. From version 5.4.5, you can use r10k to fetch Puppet environments and modules from an Artifactory Puppet repository for deployment.

To configure r10k to fetch modules from Artifactory, add the following to your `r10k.yaml` file:

```
forge:
  baseurl:
  'http://<ARTIFACTORY_HOST_NAME>:<ARTIFACTORY_PORT>/artifactory/api/puppet/<REPO_KEY>'
```

For example:

```
forge:  
  baseurl: 'http://localhost:8080/artifactory/api/puppet/puppet-local'
```

To fetch and install the Puppet modules from Artifactory, run the following command:

```
r10k puppetfile install
```

Puppet Publish (Deploying Modules)

Setting Your Credentials

To support authentication, you need to add your Artifactory credentials to your `puppet.conf` file:

Note: your credentials should be formatted as `username:password` as a [Base64](#) encoded strings

Your Artifactory credentials, formatted `username:password` as [Base64](#) encoded strings.

For example:

```
[main]  
module_repository=http://admin:AP7eCk6M6JokQpjXbkGytt7r4sf@localhost:808  
0/artifactory/api/puppet/puppet-local
```

Make sure you have an Artifactory user in order to publish modules.

Deploying Your Modules

There are two ways to deploy packages to a local repository:

1. **Using the Artifactory UI**

Once you have created your Puppet repository, you can select it in the Tree Browser and click **Deploy** to upload Puppet module. Select your module(s), and click **Deploy**.

Deploy
×

Target Repository

puppet-local
▼

Package Type: ⚠ Puppet

Repository Layout:

`[orgPath]/[module]/[orgPath]-[module]-[baseRev].tar.gz`

Type: Single | Multi

↑
Drop file here or Select file

Deploy

2. Using Artifactory REST API

For Example:

```
curl -uadmin:AP7eCk6M6JokQpjXbkGytt7r4sf -XPUT http://localhost:8080/artifactory/puppet-local/<TAR_GET_FILE_PATH> -T <PATH_TO_FILE>
```

Working with Artifactory without Anonymous Access

By default, Artifactory allows anonymous access to Puppet repositories. This is defined in the **Admin** module under **Security | General**. For details please refer to [Allow Anonymous Access](#).

To be able to trace how users interact with your repositories, you need to uncheck the **Allow Anonymous Access** setting. This means that users will be required to enter their username and password as described in [Setting Your Credentials](#) above.

Puppet Search

Artifactory supports a variety of ways to search for artifacts. For details, please refer to [Searching Artifacts](#).

Artifactory also supports, the `puppet module search [search terms ...]` command. However, a module may not be available immediately after being published, for the following reasons:

- When publishing modules to a local repository, Artifactory calculates the search index asynchronously and will wait for indexing the newly published module.
- Since a virtual repository may contain local repositories, a newly published package may not be available immediately for the same reason.
- In the case of remote repositories, a new package will only be found once Artifactory checks for it according to the [Retrieval Cache Period](#) setting.



Artifactory annotates each deployed or cached Puppet module with two properties: `puppet.name` and `puppet.version`

You can use [Property Search](#) to search for Puppet packages according to their name or version.

Cleaning Up the Local Puppet Cache

The Puppet client saves caches of modules that were downloaded, as well as their JSON metadata responses (called `.cache.json`).

The JSON metadata cache files contain the Puppet modules metadata.

We recommend removing the Puppet caches, both modules and metadata, before using Artifactory for the first time. This is to ensure that your caches only contain elements that are due to requests from Artifactory and not directly from <https://forge.puppet.com>.

Viewing Individual Puppet Module Information

Artifactory lets you view selected Puppet module metadata directly from the UI.

Drill down in the **Tree Browser** and select the **tar.gz** file you want to inspect, and view the metadata in the **Puppet Info** tab.

The screenshot shows the Artifactory interface for a file named `bfraser-grafana-2.5.0.tar.gz`. The file is selected in the **Tree Browser**. The **Puppet Info** tab is active, displaying the following **Module Info**:

Name:	bfraser-grafana
Version:	2.5.0
License:	Apache 2.0
Keywords:	grafana, graphite, influxdb, monitoring

Using Puppet 4.9.1 and Below

Up till version 4.9.1, the Puppet client does not support context path for remote Puppet Forge repositories. Therefore, we recommend using Artifactory with Puppet 4.9.2 and above.

If you need to use Puppet 4.9.1 and below you can use a workaround which uses NGINX or Apache to rewrite all requests from `/v3/*` to `/artifactory/api/puppet/<repo-name>/v3/*`.

For example, if you have a repository called `puppet-virtual`, and you are using Puppet 3.0, you would configure your proxy server to rewrite `/v3/*` to `/artifactory/api/puppet/puppet-virtual/v3/*`.

The following sections show sample configurations for NGINX and Apache for both the ports method and the sub-domain method to use a virtual repository named `puppet-virtual`.

▼ [Sample NGINX configuration using the Ports method](#)

```

## server configuration
server {
    listen 8001 ;
    location ^~/v3 {
        rewrite ^/v3/(.*) /artifactory/api/puppet/puppet-virtual/v3/$1
    }
    break;
    proxy_redirect off;
    proxy_pass http://localhost:8080/artifactory/;
}

```

▼ Sample NGINX configuration using the Subdomain method

```

## server configuration
server {
    listen 443 ssl;
    listen 80 ;
    server_name ~(?<repo>.+)\.artifactory-cluster artifactory-cluster;

    if ($http_x_forwarded_proto = '') {
        set $http_x_forwarded_proto $scheme;
    }
    ## Application specific logs
    ## access_log /var/log/nginx/artifactory-cluster-access.log
    timing;
    ## error_log /var/log/nginx/artifactory-cluster-error.log;
    rewrite ^/$ /artifactory/webapp/ redirect;
    rewrite ^/artifactory/?(/webapp)?$ /artifactory/webapp/ redirect;
    rewrite ^/(v1|v2)/(.*) /artifactory/api/docker/$repo/$1/$2;
    rewrite ^/v3/(.*) /artifactory/api/puppet/$repo/v3/$1;
    chunked_transfer_encoding on;
    client_max_body_size 0;
    location /artifactory/ {
        proxy_read_timeout 900;
        proxy_pass_header Server;
        proxy_cookie_path ~*^/. * /;
        if ( $request_uri ~ ^/artifactory/(.*)$ ) {
            proxy_pass http://artifactory/artifactory/$1;
        }
        proxy_pass http://artifactory/artifactory/;
        proxy_next_upstream http_503 non_idempotent;
        proxy_set_header X-Artifactory-Override-Base-Url
    }
    $http_x_forwarded_proto://$host:$server_port/artifactory;
    proxy_set_header X-Forwarded-Port $server_port;
    proxy_set_header X-Forwarded-Proto $http_x_forwarded_proto;
    proxy_set_header Host $http_host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}
}

```

▼ Sample Apache HTTP server configuration using the Ports method

```
#####
## this configuration was generated by JFrog Artifactory ##
#####

## add HA entries when ha is configured
<Proxy balancer://artifactory>
    BalancerMember http://10.6.16.125:8080 route=14901314097097
ProxySet lbmethod=byrequests
ProxySet stickysession=ROUTEID
</Proxy>
<VirtualHost *:80>
    ProxyPreserveHost On
    ServerName artifactory-cluster
    ServerAlias *.artifactory-cluster
    ServerAdmin server@admin

    ## Application specific logs
    ## ErrorLog ${APACHE_LOG_DIR}/artifactory-cluster-error.log
    ## CustomLog ${APACHE_LOG_DIR}/artifactory-cluster-access.log
combined
    AllowEncodedSlashes On
    RewriteEngine on
    RewriteCond %{SERVER_PORT} (.*)
    RewriteRule (.*) - [E=my_server_port:%1]
    ## NOTE: The 'REQUEST_SCHEME' Header is supported only from
apache version 2.4 and above
    RewriteCond %{REQUEST_SCHEME} (.*)
    RewriteRule (.*) - [E=my_scheme:%1]

    RewriteCond %{HTTP_HOST} (.*)
    RewriteRule (.*) - [E=my_custom_host:%1]

    RewriteRule ^/$ /artifactory/webapp/ [R,L]
    RewriteRule ^/artifactory(/)?$ /artifactory/webapp/ [R,L]
    RewriteRule ^/artifactory/webapp$ /artifactory/webapp/ [R,L]

    RequestHeader set Host %{my_custom_host}e
    RequestHeader set X-Forwarded-Port %{my_server_port}e
    ## NOTE: {my_scheme} requires a module which is supported only
from apache version 2.4 and above
    RequestHeader set X-Forwarded-Proto %{my_scheme}e
    RequestHeader set X-Artifactory-Override-Base-Url
%{my_scheme}e://artifactory-cluster:%{my_server_port}e/artifactory
    ProxyPassReverseCookiePath /artifactory /artifactory
```

```
ProxyRequests off
ProxyPreserveHost on
ProxyPass /artifactory/ balancer://artifactory/artifactory/
ProxyPassReverse /artifactory/ balancer://artifactory/artifactory/
Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e;
path=/artifactory/" env=BALANCER_ROUTE_CHANGED
</VirtualHost>
```

```
Listen 8001
```

```
<VirtualHost *:8001>
    ProxyPreserveHost On
    ServerName artifactory-cluster
    ServerAlias *.artifactory-cluster
    ServerAdmin server@admin
    ## Application specific logs
    ## ErrorLog ${APACHE_LOG_DIR}/artifactory-cluster-error.log
    ## CustomLog ${APACHE_LOG_DIR}/artifactory-cluster-access.log
    combined
```

```
    AllowEncodedSlashes On
    RewriteEngine on
```

```
    RewriteCond %{SERVER_PORT} (.*)
    RewriteRule (.*) - [E=my_server_port:%1]
    ## NOTE: The 'REQUEST_SCHEME' Header is supported only from
    apache version 2.4 and above
    RewriteCond %{REQUEST_SCHEME} (.*)
    RewriteRule (.*) - [E=my_scheme:%1]
```

```
    RewriteCond %{HTTP_HOST} (.*)
    RewriteRule (.*) - [E=my_custom_host:%1]
    RewriteRule "^/v3/(.*)$"
"/artifactory/api/puppet/puppet-virtual/v3/$1" [P]
```

```
    RewriteRule ^/$ /artifactory/webapp/ [R,L]
    RewriteRule ^/artifactory(/)?$ /artifactory/webapp/ [R,L]
    RewriteRule ^/artifactory/webapp$ /artifactory/webapp/ [R,L]
```

```
    RequestHeader set Host %{my_custom_host}e
    RequestHeader set X-Forwarded-Port %{my_server_port}e
    ## NOTE: {my_scheme} requires a module which is supported only
    from apache version 2.4 and above
    RequestHeader set X-Forwarded-Proto %{my_scheme}e
    RequestHeader set X-Artifactory-Override-Base-Url
    %{my_scheme}e://artifactory-cluster:%{my_server_port}e/artifactory
    ProxyPassReverseCookiePath /artifactory /artifactory
```

```
ProxyPass /artifactory/ balancer://artifactory/artifactory/  
ProxyPassReverse /artifactory/ balancer://artifactory/artifactory/
```



```
Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e;  
path=/artifactory/" env=BALANCER_ROUTE_CHANGED  
</VirtualHost>
```

▼ Sample Apache HTTP server configuration using the Subdomain method:

```
#####  
## this configuration was generated by JFrog Artifactory ##  
#####  
  
## add HA entries when ha is configured  
<Proxy balancer://artifactory>  
    BalancerMember http://10.6.16.125:8080 route=14901314097097  
ProxySet lbmethod=byrequests  
ProxySet stickysession=ROUTEID  
</Proxy>  
<VirtualHost *:80>  
    ProxyPreserveHost On  
    ServerName artifactory-cluster  
    ServerAlias *.artifactory-cluster  
    ServerAdmin server@admin  
  
    ## Application specific logs  
    ## ErrorLog ${APACHE_LOG_DIR}/artifactory-cluster-error.log  
    ## CustomLog ${APACHE_LOG_DIR}/artifactory-cluster-access.log  
    combined  
  
    AllowEncodedSlashes On  
    RewriteEngine on  
  
    RewriteCond %{SERVER_PORT} (.*)  
    RewriteRule (.*) - [E=my_server_port:%1]  
    ## NOTE: The 'REQUEST_SCHEME' Header is supported only from  
    apache version 2.4 and above  
    RewriteCond %{REQUEST_SCHEME} (.*)  
    RewriteRule (.*) - [E=my_scheme:%1]  
  
    RewriteCond %{HTTP_HOST} (.*)  
    RewriteRule (.*) - [E=my_custom_host:%1]  
  
    RewriteCond "%{REQUEST_URI}" "^(v1|v2|  
    )/"  
    RewriteCond "%{HTTP_HOST}" "^(.*)\.artifactory-cluster$"  
    RewriteRule "^/v3/(.*)$" "/artifactory/api/puppet/%1/v3/$1" [PT]  
    RewriteRule "^(v1|v2)/(.*)$" "/artifactory/api/docker/%1/$1/$2"
```

[PT]

```
RewriteRule ^/$ /artifactory/webapp/ [R,L]
RewriteRule ^/artifactory(/)?$ /artifactory/webapp/ [R,L]
RewriteRule ^/artifactory/webapp$ /artifactory/webapp/ [R,L]

RequestHeader set Host %{my_custom_host}e
RequestHeader set X-Forwarded-Port %{my_server_port}e
## NOTE: {my_scheme} requires a module which is supported only
from apache version 2.4 and above
RequestHeader set X-Forwarded-Proto %{my_scheme}e
RequestHeader set X-Artifactory-Override-Base-Url
%{my_scheme}e://artifactory-cluster:%{my_server_port}e/artifactory
ProxyPassReverseCookiePath /artifactory /artifactory

ProxyRequests off
ProxyPreserveHost on
ProxyPass /artifactory/ balancer://artifactory/artifactory/
ProxyPassReverse /artifactory/ balancer://artifactory/artifactory/
```

```
Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e;  
path=/artifactory/" env=BALANCER_ROUTE_CHANGED  
</VirtualHost>
```

Once you have your reverse proxy configured, you can install modules from Artifactory using the following commands:

Ports Method

```
puppet module install --module_repository http://localhost:8001  
puppetlabs-apache
```

Subdomain Method

```
puppet module install --module_repository  
http://puppet-virtual.artifactory-cluster puppetlabs-apache
```

REST API

The following REST API endpoints are available to facilitate automation for configuration management with Puppet. Artifactory also uses these endpoints to support the `librarian-puppet` and `r10k` clients:

Get Puppet Modules

Returns a list of all Puppet modules hosted by the specified repository.

For details, please refer to [Get Puppet Modules](#) in the Artifactory REST API documentation.

Get Puppet Module

Returns information about a specific Puppet module

For details, please refer to [Get Puppet Module](#) in the Artifactory REST API documentation.

Get Puppet Releases

Returns a list of all Puppet releases hosted by the specified repository.

For details, please refer to [Get Puppet Releases](#) in the Artifactory REST API documentation.

Get Puppet Release

Returns information about the specific Puppet module's release.

For details, please refer to [Get Puppet Release](#) in the Artifactory REST API documentation.

PyPI Repositories

Overview

Artifactory fully supports PyPI repositories providing:

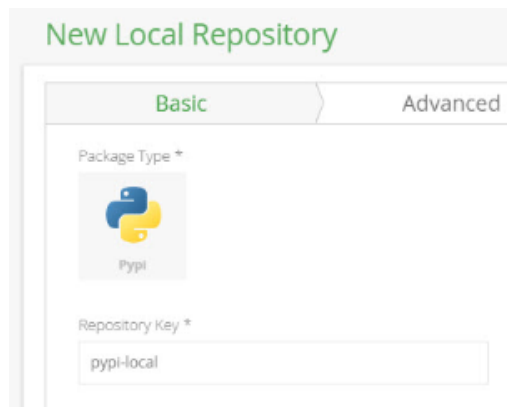
1. The ability to provision PyPI packages from Artifactory to the `pip` command line tool from all repository types.
2. Calculation of Metadata for PyPI packages hosted in Artifactory's local repositories.
3. Access to remote PyPI repositories (such as <https://pypi.python.org/>) through [Remote Repositories](#) which provide proxy and caching functionality.

4. The ability to access multiple PyPI repositories from a single URL by aggregating them under a [Virtual Repository](#).
5. Compatibility with the [setuptools](#) and its predecessor [distutils](#) libraries for uploading PyPI packages.

Configuration

Local Repositories

To create a new PyPI local repository, in the [New Local Repository](#) screen, set the **Package Type** to **PyPI**.



Page Contents

- Overview
- Configuration
 - Local Repositories
 - Remote Repositories
 - Virtual Repositories
- Resolving from Artifactory Using PIP
 - Specifying the Repository on the Command Line
 - Using Credentials
 - Using a Configuration File
 - Using a Requirements File
- Publishing to Artifactory
 - Using distutils or setuptools
 - Create the \$HOME/.pypirc File
 - Uploading
 - Publishing Manually Using the Web UI or REST
- Searching for PyPI Packages
 - Using PIP
 - Artifactory

Search

- Viewing Metadata of PyPI Packages
- Watch the Screencast

Remote Repositories

A **Remote Repository** defined in Artifactory serves as a caching proxy for a registry managed at a remote URL such as <https://pypi.python.org/>.

Artifacts (such as .whl files) requested from a remote repository are cached on demand. You can remove downloaded artifacts from the remote repository cache, however you can not manually deploy artifacts to a remote PyPI repository.

To create a repository to proxy a remote PyPI repository follow the steps below:

1. In the **Admin** module under **Repositories | Remote**, select "New"
2. Set the **Package Type** to **PyPI**, enter the **Repository Key** value, and specify the URL to the remote repository in the **URL** field as displayed below:

The screenshot shows the 'New Remote Repository' configuration interface. The 'Basic' tab is selected. The 'Package Type' is set to 'Pypi'. The 'Repository Key' is 'pypi-remote' and the 'URL' is 'https://pypi.python.org'. A 'Test' button is present.

PyPI remote repository URL

You should not include `/pypi` or `/simple` in the the PyPI remote repository URL. These suffixes are added by Artifactory when accessing the remote repository.

If you use a custom PyPI remote repository, you need to make sure it has a simple index (directory listing style) accessible by `<URL>/simple`.

3. Click "Save & Finish"

Remote Artifactory

If the remote repository is also managed by an Artifactory server, then you need to point to its PyPI API URL, for example `http://my.remote.artifactory/artifactory/api/pypi/python-project`

Virtual Repositories


A Virtual Repository defined in Artifactory aggregates packages from both local and remote repositories.

This allows you to access both locally hosted PyPI packages and remote proxied PyPI repositories from a single URL defined for the virtual repository.

To define a virtual PyPI repository, create [virtual repository](#), set its **Package Type** to be PyPI, select the underlying local and remote PyPI repositories to include in the **Basic** settings tab, click "Save & Finish".

New Virtual Repository

Basic | Advanced

Package Type *
 PyPI

Repository Key *
 pypi-virtual

General

Repository Layout
 simple-default

Public Description
 Internal Description

Include Pattern
 **/*

Exclude Pattern

Repositories

Filter...

Available Repositories

Selected Repositories

- pypi-local
- pypi-remote

Included Repositories


- pypi-local
- pypi-remote

Resolving from Artifactory Using PIP

To install the `pip` command line tool refer to [pip documentation pages](#). We recommend using `virtualenv` to separate your environment when installing PIP.

To display code snippets you can use to configure `pip` and `setup.py` to use your PyPI repository, select the repository and then click **Set Me Up**.

Set Me Up

Tool
 PyPI

Repository
 pypi-local

Deploy

To deploy package using `setuptools` you will need to add an Artifactory repository to the `.pypirc` file (usually located in your home directory)

```

1 [distutils]
2 index-servers = local
3 [local]
4 repository: http://10.100.1.110:8081/artifactory/api/pypi/pypi-local
5 username: <USERNAME>
6 password: <PASSWORD>

```

To deploy a python egg to Artifactory after changing the `.pypirc` file run the following command

```

1 python setup.py sdist upload -r <LOCAL>

```

where is the index server you defined in `.pypirc`

Resolve

To resolve packages using `pip`, add the following to `~/pip/bin.conf`

Specifying the Repository on the Command Line

Index URL

When accessing a PyPI repository through Artifactory, the repository URL must be prefixed with **api/pypi** in the path. This applies to all `pip` commands and `distutils` URLs including `pip install`.

When using `pip` to resolve PyPI packages it must point to `<Artifactory URL>/api/pypi/<repository key>/simple`.

For example, if you are using Artifactory standalone or as a local service, you would access your PyPI repositories using the following URL:

```
http://localhost:8081/artifactory/api/pypi/<repository key>/simple
```

Or, if you are using Artifactory SaaS, the URL would be:

```
https://<server name>.jfrog.io/<server name>/api/pypi/<repository key>/simple
```

Once pip is installed, it can be used to specify the URL of the repository from which to resolve:

Installing with full repository URL

```
$ pip install frog-bar -i  
http://localhost:8081/artifactory/api/pypi/pypi-local/simple
```

Using Credentials

Due to its design, pip does not support reading credentials from a file. Credentials can be supplied as part of the URL, for example `http://<username>:<password>@localhost:8081/artifactory/api/pypi/pypi-local/simple`. The password can be omitted (with the preceding colon), and in this case, the user will be prompted to enter credentials interactively.

Using a Configuration File

Aliases for different repositories can be specified through a pip configuration file, `~/.pip/pip.conf`. The file contains configuration parameters per repository, for example:

~/.pip/pip.conf

```
[global]  
index-url =  
http://user:password@localhost:8081/artifactory/api/pypi/pypi-virtual/simple
```

For more information, please refer to [PIP User Guide](#).

Using a Requirements File

A requirements file contains a list of packages to install. Usually these are dependencies for the current package. It can be created manually or using the `pip freeze` command. The index URL can be specified in the first line of the file, For example:

requirements.txt

```
--index-url http://localhost:8081/artifactory/api/pypi/pypi-local/simple  
PyYAML==3.11  
argparse==1.2.1  
frog-bar==0.2  
frog-fu==0.2a  
nltk==2.0.4  
wsgiref==0.1.2
```

Using distutils or setuptools

setuptools vs. distutils and python versions

Artifactory is agnostic to whether you use `setuptools` or `distutils`, and also to the version or implementation of Python your project uses.

The following instructions were written for Python 2.7 and `setuptools` in mind. Using different version of Python, or different tools such as `zest`, `distutils` and others may require minor modification to the instructions below.

Uploading to Artifactory using a `setup.py` script is supported in a similar way to uploading to PyPI. First, you need to add Artifactory as an index server for your user.

For instructions on using `setuptools` to package Python projects and create a `setup.py` script, please refer to the [setuptools documentation](#) and [this tutorial project](#).

Create the `$HOME/.pypirc` File

To upload to Artifactory, an entry for each repository needs to be made in `$HOME/.pypirc` as follows:

```
[distutils]
index-servers =
    local
    pypi

[pypi]
repository: https://pypi.python.org/pypi
username: mrBagthrope
password: notToBeSeen

[local]
repository: http://localhost:8081/artifactory/api/pypi/pypi-local
username: admin
password: password
```

Notice that the URL does not end with `/simple`.

The HOME environment variable

`setuptools` requires that the `.pypirc` file be found under `$HOME/.pypirc`, using the `HOME` environment variable.

On unix-like systems this is usually set by your system to `/home/yourusername/` but in certain environments such as build servers you will have to set it manually.

On Windows it must be set manually.

Uploading

After creating a `.pypirc` file and a `setup.py` script at the root of your project, you can upload your egg (tar.gz) packages as follows:

```
~/python_project $ python setup.py sdist upload -r local
```

If you are using wheel (whl) you can upload your packaged as follows:

```
~/python_project $ python setup.py bdist_wheel upload -r local
```

Or if you wish to use both egg (tar.gz) and wheel (whl), you can upload them as follows:


```
~/python_project $ python setup.py sdist bdist_wheel upload -r local
```

Where **local** is the name of the section in your `.pypirc` file that points to your Artifactory PyPI repository.

Default upload

By default, both `setuptools` and `distutils` will upload to `https://pypi.python.org/pypi` if no repository is specified.

The 'register' command should be omitted

When uploading directly to `pypi.python.org`, the documentation states that your package must first be registered by calling `python setup.py register`.

When uploading to Artifactory this is neither required nor supported and **should be omitted**.

Publishing Manually Using the Web UI or REST

PyPI packages can also be uploaded manually using the [Web UI](#) or the [Artifactory REST API](#). For Artifactory to handle those packages correctly as PyPI packages they must be uploaded with **pypi.name** and **pypi.version Properties**.

Automatic extraction of properties

While indexing the newly uploaded packages Artifactory will automatically try to extract required properties from the package metadata saved in the file. Note that not all supported files can be extracted.

Currently, only `zip`, `tar`, `tgz`, `tar.gz`, `tar.bz2`, `egg` and `whl` files can be extracted for metadata.

In addition, indexing starts after a 60 second quiet period, counting from the last upload to the current repository.

Searching for PyPI Packages

Using PIP

Artifactory supports search using `pip`'s `search` command in local, remote and virtual repositories. For example:

```
pip search
-----
$ pip search frog-fu --index
http://localhost:8081/artifactory/api/pypi/pypi-virtual/
frog-fu - 0.2a
INSTALLED: 0.2a (latest)

$ pip search irbench --index
http://localhost:8081/artifactory/api/pypi/pypi-virtual/
irbench - Image Retrieval Benchmark.
```

In this example **frog-fu** is a locally installed package, while **irbench** is found at `pypi.python.org`, both repositories aggregated by the `pypi-virtual` repository.

Specifying the index

When using the search command, the index should be specified explicitly (without the `/simple` at the end), as `pip` will ignore the **index-url** variable in its `pip.conf` file.

Artifactory Search

PyPI packages can also be searched for using Artifactory's [Property Search](#). All PyPI packages have the properties **pypi.name**, **pypi.version** and **pypi.summary** set by the uploading client, or later during indexing for supported file types.

Viewing Metadata of PyPI Packages

Artifactory lets you view selected metadata of a PyPI package directly from the UI.

In the **Artifacts** module **Tree Browser**, drill down to select the file you want to inspect. The metadata is displayed in the **PyPI Info** tab.

geokey-0.6.tar.gz Download Actions

General **PyPI Info** Effective Permissions Properties Watchers Builds Governance

Package Info

Name:	geokey
Version:	0.6
Platform:	UNKNOWN
Summary:	Open-source participatory mapping framework
Keywords:	
Homepage:	http://geokey.org.uk
Download URL:	https://github.com/excites/geokey/releases
Author:	Oliver Roick
Author Email:	o.roick@ucl.ac.uk
License:	Apache 2.0

Watch the Screencast

RubyGems Repositories

Overview

Artifactory provides full support for RubyGems repositories including:

- Local repositories with RubyGems API support
- Caching and proxying remote RubyGems repositories
- Virtual repositories that aggregate multiple local and remote repositories including indices of both gems and specifications
- Support for common Ruby tools such as gem and bundler

For general information on configuring Artifactory to work with RubyGems, please refer to [Configuring Repositories](#).

Page Contents

- Overview
- General Configuration
- Local Repositories
 - Usage
- Remote Repositories
 - Usage
- Virtual Repositories
 - Usage
- Using the REST API
- Viewing RubyGems Artifact Information
 - Viewing and Extracting License Information
- Watch the Screencast

General Configuration

All RubyGems repositories must be prefixed with `api/gems` in the path

When using the RubyGems command line to access a repository through Artifactory, the repository URL must be prefixed with `api/gems` in the path.

All RubyGems commands, including `gem source` and `gem push`, must prepend the repository path with `api/gems`.

For example, if you are using Artifactory standalone or as a local service, you would access your RubyGems repositories using the following URL:

```
http://localhost:8081/artifactory/api/gems/<repository key>
```

Or, if you are using Artifactory SaaS, the URL would be:

```
https://<server name>.jfrog.io/<server name>/api/gems/<repository key>
```

Using RubyGems repositories with Artifactory version 3.4.1 and below, and Java 7 update 40 or higher

If you are using RubyGems repositories with Java 7 update 40 or higher, you may receive the following exception:

```
org.jruby.exceptions.RaiseException: (SystemStackError) stack level too deep  
at ...
```

This is due to an [issue](#) with Artifactory's use of JRuby.

If you are using Artifactory version 3.4.1 and below, you need to define the following [System Property](#) and restart Artifactory:

```
jruby.compile.invokedynamic=false
```

In Artifactory version 3.4.2, an enhancement has been implemented to overcome this issue and from this version on, no action is required.

Local Repositories

Local RubyGems repositories are physical, locally-managed repositories into which you can deploy and manage your in-house Gems.

To create a local RubyGems repository, in the **Admin** module, under **Repositories | Local**, click "New" and set **RubyGems** to be the **Package Type**.



You can set up a local RubyGems repository as follows:

You need to add the repository source URL by modifying your `~/.gemrc` file or using the `gem source` command as displayed below. You can extract the source URL by selecting the repository in the [Tree Browser](#) and clicking **Set Me Up**.

Set Me Up

Tool

Gems

Repository

gem1

General

In order for your client to upload and download Gems from Artifactory repository you will need to add it to your `~/.gemrc` file using the following command

```
1 gem source -a http://<USERNAME>:<API_KEY>@http://10.100.1.110:8081/artifactory/api/gems/gem1/
```

In case anonymous access is enabled you can also use the following

```
1 gem source -a http://10.100.1.110:8081/artifactory/api/gems/gem1/
```

You can view the order of resolution by running

```
1 gem source --list
```

Make sure that Artifactory repository is at the top of the list. It is recommended to use virtual repository to encapsulate both local and remote repositories under a single URL.

If you want to setup the credentials for your gem tool either include your `API_KEY` in the `~/.gems/credentials`, or run the

Notice that there are two sources. First, the remote proxy, then the local one. This will effectively allow you to retrieve Gems from both of them in the specified order.

All RubyGems repositories must be prefixed with `api/gems` in the path

When using the RubyGems command line to access a repository through Artifactory, the repository URL must be prepended with `api/gems` in the path.

```
gem source -a http://localhost:8081/artifactory/api/gems/my-gems-local/
```

Usage

First, setup the appropriate **credentials** for the **gem** tool: either include the API key in the `~/.gem/credentials` file or issue this command:

Setting Up Credentials

```
curl http://localhost:8081/artifactory/api/gems/<repository>/api/v1/api_key.yaml -u admin:password > ~/.gem/credentials
```

Running on Linux

You may need to change the permissions of the credentials file to 600 (`chmod 600`)

Running on Windows

The credentials file is located under `%USERPROFILE%/.gem/credentials`

You also need to set the API key encoding to be "ASCII". For example:

```
curl http://localhost:8081/artifactory/api/gems/<repository key>/api/v1/api_key.yaml | Out-File ~/.gem/credentials -Encoding "ASCII"
```

API keys

You can modify the credentials file manually and add different API keys. You can later use `gem push -k key` to choose the relevant API key.

In order to **push** gems to the local repository, you can set the global variable `$RUBYGEMS_HOST` to point to the local repository as follows:

Setting RUBYGEMS_HOST

```
export RUBYGEMS_HOST=http://localhost:8081/artifactory/api/gems/<repository key>
```

To get this value, select your repository in the [Tree Browser](#) and click **Set Me Up**.

Deploy

In order to push gems to this repository, you can set the global variable `$RUBYGEMS_HOST` to point to it as follows:

```
1 export RUBYGEMS_HOST=http://localhost:8080/artifactory/api/gems/gems-local
```

You can also specify the target repository when pushing the gem by using the `--host` option:

```
1 gem push <PACKAGE> --host http://localhost:8080/artifactory/api/gems/gems-local
```

Alternatively you could use the `gem push` command with `--host`, and optionally, `--key` to specify the relevant API key.

Make sure you are familiar with your effective sources and their order as specified in the `~/.gemrc` file.

Also, make sure you are familiar with your global `$RUBYGEMS_HOST` variable before you issue a `gem push` command or use the `push --host` option.

When a local repository is first created, it is populated with `rubygems-update-*.gem` by default. In order to disable this behavior, you must change the [System Properties](#) to include:

```
artifactory.gems.gemsAfterRepoInitHack=false
```

Make sure you deploy to a "gems" folder

When deploying Gems to your repositories, you need to place them in a `gems` folder for Artifactory to include them in its indexing calculations.

Remote Repositories

A remote RubyGems repository serves as a caching proxy for a repository managed at a remote URL such as <http://rubygems.org>.

Once requested, artifacts (Gems) are cached on demand. They can then be removed, but you cannot manually deploy anything into a remote repository.

To create a remote RubyGems repository, execute the following steps:

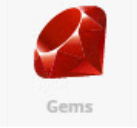
1. in the **Admin** module, under **Repositories | Remote**, click "New" and set **RubyGems** to be the **Package Type**.

2. Set the **Repository Key**, and specify the **URL** to the remote repository. The example below references rubygems.org.

New Remote Repository

Basic Advanced Replications

Package Type *

 Gems

Repository Key *

URL *


Usage

In order to allow the integration with the **gem** command line tool, you must add the relevant source URL to your RubyGems configuration.

1. In the **Tree Browser**, select your newly created repository and click **Set Me Up**.
2. Copy the source URL from the **RubyGems Sources** section.

Set Me Up

Tool

 Gems

Repository

gem1

General

In order for your client to upload and download Gems from Artifactory repository you will need the it to your `~/.gemrc` file using the following command

```
1 gem source -a http://<USERNAME>:<API_KEY>@http://10.100.1.110:8081/artifactory/api/gems/gem1/
```

In case anonymous access is enabled you can also use the following

```
1 gem source -a http://10.100.1.110:8081/artifactory/api/gems/gem1/
```

You can view the order of resolution by running

```
1 gem source --list
```

Make sure that Artifactory repository is at the top of the list. It is recommended to use virtual repository to encapsulate both local and remote repositories under a single URL.

If you want to setup the credentials for your gem tool either include your `API_KEY` in the `~/.gems/credentials`, or run the

3. Add this URL by modifying your `~/.gemrc` file or using the `gem source` command as follows:

All RubyGems repositories must be prefixed with `api/gems` in the path

When using the RubyGems command line to access a repository through Artifactory, the repository URL must be prefixed with `api/gems` in the path.

```
gem source -a http://localhost:8081/artifactory/api/gems/rubygems/
```

Additional Gem Commands You Can Use

You can remove previous source entries by modifying your `~/.gemrc` file manually or by running `gem sources -r`. You can also run `gem sources --list` to know what your effective sources are and their order.

Overcoming Unauthorized Status Failures

Some `gem/bundler` commands may fail with an Unauthorized (401) status. In some cases these can be overcome by using one of the following options:

- Enable the "Anonymous User" by checking **Allow Anonymous Access** in **Security General Configuration** as described in [Managing Users](#).
- Create a specialized [Permission Target](#) that allows anonymous access only to the remote repository.
- Use a source URL with embedded credentials, such as: `gem sources -a http://user:password@host/...`

Virtual Repositories

A Virtual RubyGems repository (or "repository group") can aggregate multiple local and remote RubyGems repositories, seamlessly exposing them under a single URL.

The repository is virtual in that you can resolve and retrieve artifacts from it but you cannot deploy anything to it. For more information please refer to [Virtual Repositories](#).

The procedure for setting up a virtual repository is very similar to setting up a local or remote repository, however as a last step, you need to select the repositories that will be included in the virtual repository you are creating.

Repositories

Available Repositories

- gem1
- ruby-ruby
- ruby1
- rubygems
- ruby-remote

Selected Repositories

- ruby-gem-local
- rubygems-remote

Included Repositories

Usage

Using a virtual RubyGems repository you can aggregate both your local and remote repositories.

You need to set the right repository source URL, in the same way as described in [Usage](#) for a local RubyGems repository, just with the appropriate repository key as follows:

source: `http://localhost:8081/artifactory/api/gems/<repository key>/`

target: `http://localhost:8081/artifactory/api/gems/<repository key> (no slash at the end!)`

Using the REST API

The REST API provides access to the Gems Add-on through the repository key using the following URL:

`http://localhost:8081/artifactory/api/gems/<repository key>/`

In addition to the basic binary repository operations, such as download, deploy, delete etc., the following RubyGems.org API Gem commands are supported:

Gem command	Curl syntax example	Remarks
Info	<code>curl http://localhost:8081/artifactory/api/gems/<repository key>/api/v1/gems/my_gem.(json xml yaml)</code>	Optionally indicate JSON / XML / YAML (default: JSON)
Search	<code>curl http://localhost:8081/artifactory/api/gems/<repository key>/api/v1/search.(json xml yaml)?query=[query]</code>	Will search for gems with name containing <i>query</i>
Dependencies	<code>curl http://localhost:8081/artifactory/api/gems/<repository key>/api/v1/dependencies?gems=[gem1,...]</code>	Use a csv of gem names for the value of <i>gems</i>
Yank	<code>curl -X DELETE http://localhost:8081/artifactory/api/gems/<repository key>/api/v1/yank</code> <code>-d 'gem_name=gem_name' -d 'version=0.0.1' -d 'platform=ruby'</code>	Deletes the specific gem file from the repository


Indexing is done automatically by Artifactory in the background, however if you still need to recreate or update the spec index files, the following REST API commands are also available:

REST command	Curl syntax example	Remarks
ReIndex	<code>curl -X POST http://localhost:8081/artifactory/api/gems/<repository key>/reindex</code>	Re-creates all spec index files.
Update index	<code>curl -X POST http://localhost:8081/artifactory/api/gems/<repository key>/updateIndex</code>	Updates all spec index files if needed.

Viewing RubyGems Artifact Information

If you select a RubyGems artifact in the Tree Browser you can select the **RubyGems** tab to view detailed information on the selected artifact.

Package Info

Name : rubygems-update
 Version : 2.0.6
 Platform : ruby
 Summary : RubyGems is a package management framework for Ruby
 Authors : Jim Weirich, Chad Fowler, Eric Hodel
 Homepage : <http://rubygems.org>
 Repository Path : ruby-gem-local:gems/rubygems-update-2.0.6.gem 
 Description : RubyGems is a package management framework for Ruby. This gem is an update for the RubyGems software. You must have an installation of RubyGems before this update can be applied. See Gem for information on RubyGems (or `ri Gem`) To upgrade to the latest RubyGems, run: \$ gem update --system # you might need to be an administrator or root See UPGRADING.rdoc for more details and alternative instructions. ---- If you don't have RubyGems installed, you can still do it manually. * Download from: <https://rubygems.org/pages/download> * Unpack into a directory and cd there * Install with: ruby setup.rb # you may need admin/root privilege For more details and other options, see: ruby setup.rb --help


Dependencies

Name	Version	Type
minitest	~> 5.0	Development
rdoc	~> 4.0	Development
builder	~> 2.1	Development
hoe-seattlerb	~> 1.2	Development
ZenTest	~> 4.5	Development
rake	~> 0.9.3	Development
hoe	~> 3.7	Development

Viewing and Extracting License Information

From version 4.4, Artifactory can scan a Gem to extract information about license files embedded in the Gem.

To scan for license files, in the **General** tab, click **SCAN**.

 **rubygems-update-2.0.6.gem** Download Actions

General RubyGems Effective Permissions Properties Watchers Builds Governance

Info

Name:	rubygems-update-2.0.6.gem
Repository Path:	gem1/gems/rubygems-update-2.0.6.gem
Module ID:	N/A
Deployed by:	_system_
Size:	332.50 KB
Created:	16-07-15 07:50:25 UTC
Last Modified:	16-07-15 07:50:25 UTC
Licenses:	Not Found Add Scan Delete
Downloaded:	2
Last Downloaded By:	admin
Last Downloaded:	23-08-15 09:04:28 UTC
Remote Downloaded:	0
Watching Since:	27-07-15 19:53:05 UTC

Filtered

Checksums

SHA-2:	Calculate
SHA-1:	317fc099c2cd764cbf4aae134a749bb59babd253 (Uploaded: Identical)
MD5:	0b0e05be407c929414ed2a99ea10982f (Uploaded: Identical)

When scanning a Gem for licenses, Artifactory searches for the following file names: *license*, *LICENSE*, *license.txt*, *LICENSE.txt*, *LICENSE.TXT*

Searching for RubyGems license files

You can override or extend the list of files Artifactory searches for by modifying the `artifactory.archive.licenseFile.names` property.

Once a license has been extracted, Artifactory annotates the Gem with a corresponding property.

rubygems-update-2.0.6.gem Download Actions

General RubyGems Effective Permissions **Properties** Watchers Builds Governance

Add: [Property](#) | Property Set

Recursive [?](#)

3 Properties

Delete Page 1 of 1

Property	Value(s)
gem.name	rubygems-update
gem.version	2.0.6
artifactory.licenses	MIT

Watch the Screencast

SBT Repositories

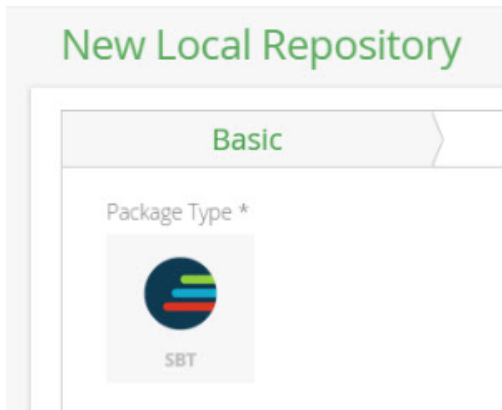
Overview

Artifactory provides integration with sbt, allowing you to configure it to resolve dependencies from, and deploy build output to sbt repositories. All you need to do is make minor modifications to your `build.sbt` configuration file.

Configuration

Local Repositories

A local sbt repository is used as a target to which you can deploy the output of your `build.sbt` script. To create an sbt repository, set the **Package Type** to **SBT**.



Page Contents

- Overview
- Configuration
 - Local Repositories
 - Remote Repositories
 - Virtual Repositories
- Configuring sbt
 - Configuring Proxy Repositories
 - Configuring Artifact Resolution
 - Deploying Artifacts
- Sample Project

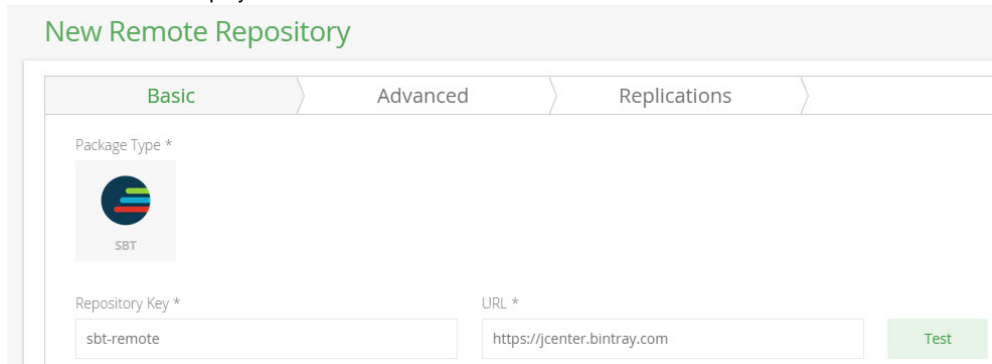
Remote Repositories

A [Remote Repository](#) defined in Artifactory serves as a caching proxy for a registry managed at a remote URL.

Artifacts (such as JAR files) requested from a remote repository are cached on demand. You can remove downloaded artifacts from the remote repository cache, however, you can not manually deploy artifacts to a remote SBT repository.

To define a remote sbt repository to proxy a remote sbt registry follow the steps below:

1. In the **Admin** module, under **Repositories | Remote**, click "New".
2. In the New Repository dialog, set the **Package Type** to **SBT**, set the **Repository Key** value, and specify the URL to the remote registry in the **URL** field as displayed below:



3. Click "Save & Finish"

The parameters needed to configure remote sbt repositories are identical to those used for Maven repositories. For more details, please refer to [Type-Specific Basic Settings](#) under [Remote Repositories](#).


Virtual Repositories

A Virtual Repository defined in Artifactory aggregates packages from both local and remote repositories.

This allows you to access both locally hosted JARS and remote proxied sbt registries from a single URL defined for the virtual repository. To define a virtual sbt repository, create a [virtual repository](#), set the **Package Type** to be sbt, and select the underlying local and remote sbt repositories to include in the **Basic** settings tab.

New Virtual Repository

Basic > Advanced

Package Type *
 SBT

Repository Key *

General

Repository Layout

Public Description

Internal Description

Include Patterns ?

Exclude Patterns ?

Repositories

Filter...

Available Repositories

- plugins-release-local
- plugins-snapshot-local
- jcenter
- libs-release
- libs-snapshot
- plugins-release
- plugins-snapshot
- remote-repos

Selected Repositories

- sbt-local
- sbt-remote

Included Repositories

- sbt-local
- sbt-remote

Cancel < Back Next > **Save & Finish**

Click "Save & Finish" to create the repository.

The parameters needed to configure virtual sbt repositories are identical to those used for Maven repositories. For more details, please refer to [Virtual Repositories](#).

Configuring sbt

To configure sbt to resolve and deploy artifacts through sbt repositories defined in Artifactory, simply select one of the sbt repositories in the Tree Browser and click **Set Me Up**. Artifactory will display code snippets you can use in the relevant sbt files.

Set Me Up

Tool

Repository

General

You can define proxy repositories in the `~/.sbt/repositories` file in the following way:

```

1 [repositories]
2 local
3 my-ivy-proxy-releases: http://localhost:8081/artifactory/sbt-local/
  [organization]/[module]/(scala_[scalaVersion]/)(sbt_[sbtVersion]/)[revision]/[type]s/[artifact](-
  [classifier]).[ext]
4 my-maven-proxy-releases: http://localhost:8081/artifactory/sbt-local/

```

In order to specify that all resolvers added in the sbt project added should be ignored in favor of those configured in the repositories configuration, add the following configuration option to the sbt launcher script:

```

1 -Dsbt.override.build.repos=true

```

You can add this setting to the `/usr/local/etc/sbtopts` file

Configuring Proxy Repositories

To configure a repository defined in Artifactory as a proxy repository for sbt, add the code snippet below to your `~/.sbt/repositories` file.

```
[repositories]
local
my-ivy-proxy-releases: http://<host>:<port>/artifactory/<repo-key>/,
[organization]/[module]/(scala_[scalaVersion]/)(sbt_[sbtVersion]/)[revision]/[type]s/[artifact](-[classifier]).[ext]
my-maven-proxy-releases: http://<host>:<port>/artifactory/<repo-key>/
```

Where `<host>:<port>` are the host URL and port on which Artifactory is running.

For example, if you are running Artifactory on your local machine, on port 8081, and want to proxy Ivy repositories through a repository called `sbt-ivy-proxy`, and proxy Maven repositories through a repository called `sbt-maven-proxy` you would use:

```
[repositories]
local
my-ivy-proxy-releases: http://localhost:8081/artifactory/sbt-ivy-proxy/,
[organization]/[module]/(scala_[scalaVersion]/)(sbt_[sbtVersion]/)[revision]/[type]s/[artifact](-[classifier]).[ext]
my-maven-proxy-releases: http://localhost:8081/artifactory/sbt-maven-proxy/
```

Proxying Maven and Ivy repositories separately

We recommend using different repositories to proxy Maven and Ivy repositories as a best practice described in [Proxying Ivy Repositories](#) in the [SBT Reference Manual](#).

To specify that all resolvers added in the sbt project should be ignored in favor of those configured in the repositories configuration, add the following configuration option to the sbt launcher script:

```
-Dsbt.override.build.repos=true
```

You can also add this setting to your `/usr/local/etc/sbtopts`

For more details on sbt proxy repositories, please refer to [Proxy Repositories](#) in the [SBT Reference Manual](#).

Configuring Artifact Resolution

To resolve artifacts through Artifactory, simply add the following code snippet to your `build.sbt` file:

```
resolvers += "Artifactory" at
"http://<host>:<port>/artifactory/<repo-key>/"
```

Where `<host>:<port>` are the host URL and port on which Artifactory is running, and `repo-key` is the Artifactory repository through which you are resolving artifacts

Deploying Artifacts

To deploy sbt build artifacts to repositories in Artifactory, add the following code snippets to your `build.sbt` file.

For **releases**, add:

```
publishTo := Some("Artifactory Realm" at
"http://<host>:<port>/artifactory/<repo-key>")
credentials += Credentials("Artifactory Realm", "<host>", "<USERNAME>",
"<PASS>")
```

For **snapshots**, add:

```
publishTo := Some("Artifactory Realm" at
"http://<host>:<port>/artifactory/<repo-key>;build.timestamp=" + new
java.util.Date().getTime)
credentials += Credentials("Artifactory Realm", "<host>", "<USERNAME>",
"<PASS>")
```

Where `<host>:<port>` are the host URL and port on which Artifactory is running, and `repo-key` is the Artifactory repository to which you are deploying artifacts.

Sample Project

A sample SBT project that uses Artifactory is available on [GitHub](#) and can be freely forked.

Vagrant Repositories

Overview

On top of [general support](#) for advanced artifact management, Artifactory support for [Vagrant](#) provides:

1. Distribution and sharing of Vagrant boxes within your organization.
2. Calculation of Metadata for Vagrant boxes hosted in Artifactory's local repositories
3. Extensive security features that give you fine-grained access control over boxes.
4. Support for flexible repository layouts that allow you to organize your boxes and assign access privileges according to projects or development teams.
5. Smart searches for boxes.

Configuration

Local Repositories

To create a local Vagrant repository to host your Vagrant boxes, create a new Local Repository and set **Vagrant** as the **Package Type**.

New Local Repository

Basic

Advanced

Package Type *



Repository Key *

vagrant-local

Page Contents

- Overview
- Configuration
 - Local Repositories
- Deploying Vagrant Boxes
 - Deploying a package using the UI
 - Set Me Up
 - Deploying a package using Matrix Parameters
 - Setting the Target Path
- Provisioning Vagrant Boxes
- Authenticated Access to Servers
- Watch the Screencast

Deploying Vagrant Boxes

Deploying a package using the UI

To deploy a Vagrant box to Artifactory, select the repository to which you want to deploy your Vagrant box and click **Deploy**..

The **Deploy** dialog is displayed with your selected repository as the **Target Repository** and a default **Target path**.

Deploy

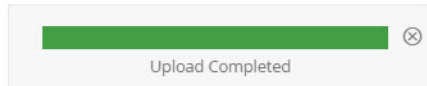


Target Repository

vagrant-local

Package Type: Vagrant

Type: **Single** | Multi



Target Path [?](#)

`/precise64-virtualbox-1.0.0.box; precise64-virtualbox-`

Deploy

You can add properties you wish to attach to your box as parameters to the target path.

For example, to upload the box **precise64-virtualbox-1.0.0.box**, and specify that its name is **precise64**, with a provider of **virtualbox** and the version is **1.0.0**, you would enter:

Specifying the Target Path

```
/precise64-virtualbox-1.0.0.box;box_name=precise64;box_provider=virtualbox  
;box_version=1.0.0
```

Set Me Up

You can also select your repository and click **Set Me Up** to view the cURL command you can use to upload your box.

Tool

Vagrant

Repository

vagrant-local

Deploy

To deploy Vagrant boxes to this Artifactory repository using an explicit URL with Matrix Parameters use:

```
1 curl -i -u<USERNAME>:<API_KEY> -T <PATH_TO_FILE> "http://10.100.1.110:8081/artifactory/vagrant-local/{vagrantBoxName.box};box_name={name};box_provider={provider};box_version={version}"
```

Resolve

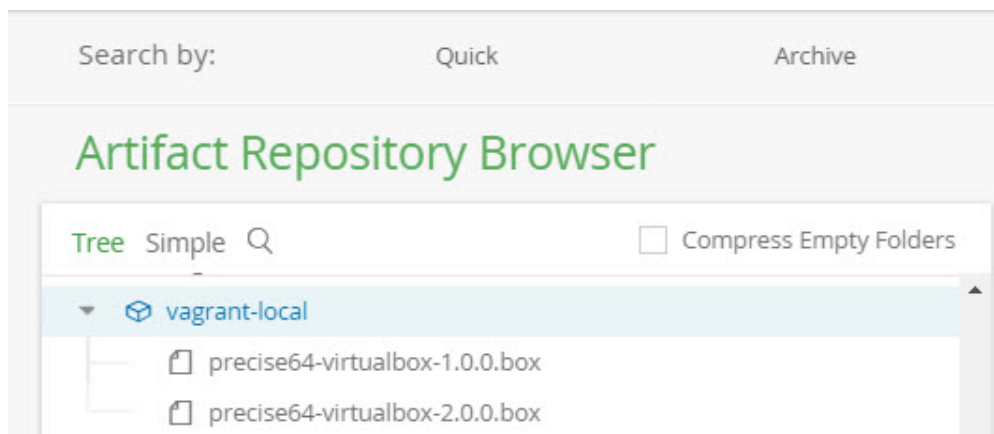
To provision a Vagrant box, all you need is to construct it's name in the following manner.

```
1 vagrant box add "http://10.100.1.110:8081/artifactory/api/vagrant/vagrant-local/{boxName}"
```

Be careful with spaces

Make sure you don't enter any superfluous spaces in the Target Path specification.

Once you have deployed your Vagrant box, and Artifactory has recalculated the repository index, your repository should be organized as displayed below:



Deploying a package using Matrix Parameters

You can also deploy Vagrant boxes to Artifactory with an explicit URL using [Matrix Parameters](#).

The URL is built similarly to the [Target Path](#) format as follows:

Deploying a package using Matrix Parameters

```
PUT "http://{Artifactory
URL}/{vagrantRepoKey}/{vagrantBoxName.box};box_name={name};box_provider={p
rovider};box_version={version}"
```

For example, to upload the box **precise64-virtualbox-1.0.0.box**, and specify that its name is **precise64**, with a provider of **virtualbox** and the version is **1.0.0**, you would enter:

Example

```
PUT
"http://localhost:8080/artifactory/vagrant-local/precise64-virtualbox-1.0.
0.box;box_name=precise64;box_provider=virtualbox;box_version=1.0.0"
```

Setting the Target Path

The **Target Path** can be anywhere in the repository, but it has to contain the 3 mandatory matrix parameters: **box_name**, **box_provider** and **box_version** and the file name must end with **.box**. The format is as follows:

Target Path Format

```
PUT "http://{Artifactory
URL}/{vagrantRepoKey}/{path/to/vagrantBoxName.box};box_name=[name];box_pro
vider=[provider];box_version=[version]"
```

name	The value to assign to the <code>box_name</code> property used to specify the Vagrant box name.
provider	The value to assign to the <code>box_provider</code> property used to specify the Vagrant box <code>provider</code> (virtualbox/lxc or others).
version	The value to assign to the <code>box_version</code> property used to specify the Vagrant box version (must comply with Vagrant's versioning schema)

Provisioning Vagrant Boxes

Vagrant boxes are available through the following URL:

Vagrant box URL

```
vagrant box add "http://{Artifactory
URL}/api/vagrant/{vagrantRepoKey}/{boxName}"
```

Specifying the path to the box

With Vagrant client commands, make sure you don't specify the path to a box in the command. The path should be specified using properties.

For example, to provision a Vagrant box called **precise64** from a repository called **vagrant-local**, you would construct it's name in the following manner:

Provisioning a Vagrant box

```
vagrant box add
"http://localhost:8080/artifactory/api/vagrant/vagrant-local/precise64"
```

You can select the repository from which you want to provision your box, and click [Set Me Up](#) to get the specific URL for the selected repository.

You can also, optionally, pass parameters to specify a specific box version or provider. For example:

Provisioning a Vagrant box by version

```
vagrant box add
"http://localhost:8080/artifactory/api/vagrant/vagrant-local/precise64
--provider virtualbox --box-version 1.0.0"
```

In addition, boxes can be provisioned using properties; this is useful when you want to download the latest box tagged by a specific property. The properties query parameter value should comply with [Using Properties in Deployment and Resolution](#).

Examples:

Provisioning a Vagrant box by version

```
vagrant box add
"http://localhost:8080/artifactory/api/vagrant/vagrant-local/precise64?prop
erties=box_version%2B=3.0.0"
```

Note the '%2B' encoding on the command for the '+' symbol (which is for Mandatory properties: key+=value)

The following example downloads a box with `box_name=trusty64`, `box_version=3.0.0` from `path="folder"`.

It uses an optional "path" property (in addition to the mandatory properties) to specify the path where the box is stored in Artifactory. We will use this property for resolution of the box.

```
vagrant box add
"http://localhost:8080/artifactory/api/vagrant/vagrant-local/trusty64?prop
erties=box_version%2B=3.0.0;path%2B=folder"
```

Note the format for resolution of multiple properties: `key1+=value1;key2+=value2...`

4 Properties

⊗ Delete < Page 1 of 1 >

Property	Value(s)
box_name	trusty64
box_version	3.0.0
box_provider	aws
path	folder

Authenticated Access to Servers

If you need to access a secured Artifactory server that requires a username and password, you need to specify 2 environment variables:

1. ATLAS_TOKEN - A Base64 encoded string of the user credentials (formatted `username:password`).
2. VAGRANT_SERVER_URL - The base URL for the Artifactory server.

Setting ATLAS_TOKEN and VAGRANT_SERVER_URL

```
export ATLAS_TOKEN={token}
export VAGRANT_SERVER_URL=http://{Artifactory
URL}/api/vagrant/{vagrantRepoKey}
For example:
export ATLAS_TOKEN=YWRtaW46QVAzWGhzWmlDU29NVmtaQ2dCZEY3XXXXXXXXX
export VAGRANT_SERVER_URL=http://localhost:8081/api/vagrant/vagrant-local
```

Getting the ATLAS_TOKEN directly from Artifactory

You can use the following command to get the ATLAS_TOKEN string directly from Artifactory:

```
$ curl -uadmin:password "http://localhost:8080/artifactory/api/vagrant/auth"
YWRtaW46QVAzWGhzWmlDU29NVmtaQ2dCZEY3XXXXXXXXX
```

Watch the Screencast

VCS Repositories

Overview

Today, many technologies that are consumed as pure source files are deployed as binaries (for example, PHP, Rails, Node.js, Ruby etc.). As a Binary Repository Manager, Artifactory completes the picture by providing you an easy, safe and secure way to consume those binaries through support for VCS repositories.

Artifactory support for VCS provides:

1. The ability to list all the tags and branches of any VCS repository.
2. Access to remote VCS repositories such as GitHub (<https://github.com>) and Bitbucket (<https://bitbucket.org>) through Remote Repositories which provide the

Page Contents

- Overview
- Configuration
 - Repository Layout
 - Remote Repositories
 - Git Providers
 - Using Stash

- usual proxy and caching functionality.
3. On-demand local caching of tags and branches for later use in case of network instability or hosted VCS service downtime.
 4. The ability to assign access privileges according to projects or development teams.

Configuration

Repository Layout

VCS repositories require an appropriate repository layout to support a more hierarchical layout of the cached items.

To use a hierarchical layout for your repository, you can either use the built-in **vcs-default** layout that comes with Artifactory out-of-the-box, or define a [Custom Layout](#). This will ensure that different maintenance features like [Version Cleanup](#) will work correctly.

- [Using the API](#)
 - [Get VCS Tags](#)
 - [Get VCS Branches](#)
 - [Download Tag](#)
 - [Download File within a Tag](#)
 - [Download Branch](#)
 - [Download File within a Branch](#)
 - [Download Release](#)
 - [Examples](#)
 - [Accessing Private VCS Repositories](#)
- or
Bitbucket

Repository layout is final

Once a remote repository is created you cannot change its layout so we recommend that you define it beforehand.

Built-in Custom Layout: vcs-default

Artifactory's built-in default layout for VCS repositories (**vcs-default**) can work with both GitHub and BitBucket.

Below is an example of a [Custom Layout](#) named `vcs-default`:

Edit vcs-default Repository Layout

Repository Layout Settings

Layout Name *

vcs-default

Artifact Path Pattern * [?](#)

[orgPath]/[module]/[refs<tags|branches>]/[baseRev]/[module]-[baseRev](-[fileItegRev])(-[classifier]).[ext]

Distinctive Descriptor Path Pattern [?](#)

Folder Integration Revision RegExp * [?](#)

.*

File Integration Revision RegExp * [?](#)

[a-zA-Z0-9]{40}

Test Artifact Path Resolution

Test Path

Test

Regular Expression View

Resolve

You can configure this Custom Layout as displayed in the image above, or simply copy the below code snippet into the relevant section in your Artifactory Central Configuration (in the **Admin** module, under **Advanced | Config Descriptor**):

```
<repoLayout>
  <name>vcs-default</name>

  <artifactPathPattern>[orgPath]/[module]/[refs<tags|branches>]/[baseRev]/
  [module]-[baseRev](-[fileItegRev])(-[classifier]).[ext]</artifactPathPat
  tern>

  <distinctiveDescriptorPathPattern>false</distinctiveDescriptorPathPatter
  n>

  <folderIntegrationRevisionRegExp>.*</folderIntegrationRevisionRegExp>

  <fileIntegrationRevisionRegExp>[a-zA-Z0-9]{40}</fileIntegrationRevisionR
  egExp>
</repoLayout>
```

If a repository package layout is in a corresponding folder hierarchy, the Artifactory Version Cleanup tool correctly detects previously installed versions.

Delete Versions



< Page 1 of 1 >

<input type="radio"/>	Group ID	Version	Directories Count
<input type="radio"/>	junit	3.8.1	1
<input type="radio"/>	junit	4.7	1

Cancel

Delete Selected

Searching for artifact versions using the REST API also works properly:

```
$ curl
"http://localhost:8081/artifactory/api/search/versions?g=jquery&a=jquery
&repos=github-cache"
{
  "results" : [ {
    "version" : "2.0.3",
    "integration" : false
  }, {
    "version" : "master-062b5267d0a3538f1f6dee3df16da536b73061ea",
    "integration" : true
  } ]
}
```

Remote Repositories

You need to create a [Remote Repository](#) which serves as a caching proxy for [github.com](#). If necessary, you can do the same for [bitbucket.org](#) or any other remote git repository that you need.

Artifacts (such as tar.gz files) requested from a remote repository are cached on demand. You can remove downloaded artifacts from the remote repository cache, however, you can not manually deploy artifacts to a remote repository.

To create a remote repository to proxy [github.com](#) or an on-prem GitHub Enterprise repository, follow the steps below:

1. In the **Admin** module, under **Repositories | Remote**, click "New" and set **VCS** to be the **Package Type**.
2. Set the **Repository Key**, and specify the **URL** to be <https://github.com> (or your GitHub Enterprise URL endpoint) as displayed below:

New Remote Repository

Basic > Advanced > Replications

Package Type *

VCS

Repository Key *

URL *

3. Under VCS Settings, select the GitHub provider in the **Git Provider** field and click "Save & Finish".

VCS Settings

Git Provider

Max Unique Snapshots ?

List Remote Folder Items ?

Git Providers

Artifactory supports proxying the following Git providers out-of-the-box: GitHub, Bitbucket, Stash, a remote Artifactory instance or a custom Git repository as displayed below:

VCS Settings

Git Provider

Select Git provider...

- GitHub
- BitBucket
- Stash
- Artifactory
- Custom

Use the custom provider if you have a Git repository which does not exist in the pre-defined list. In this case, you need to provide Artifactory with the download paths for your Git tarballs.

You do so by providing 4 placeholders:

Placeholder	Description
{0}	Identifies the username or organization name.
{1}	Identifies the repository name.
{2}	Identifies the branch or tag name.

{3}	Identifies the file extension to download.
-----	--

For example, GitHub exposes tarball downloads at: `https://github.com/<user>/<repo>/archive/<tag/branch>.<extension>`

Therefore, the custom download path configured for Artifactory should be `{0}/{1}/archive/{2}.{3}`

Using Stash or Bitbucket

If you are using Stash or BitBucket, you need to download and install the [BitBucket Server Archive Plugin](#).

Once the JAR is downloaded, select it in your Stash UI, under **Administration | Manage add-ons | Upload add-on**.

Once you have installed the add-on you need to restart Stash.

Using the API

VCS repositories must be prefixed with `api/vcs` in the path

When accessing a VCS repository through Artifactory, the repository URL must be prefixed with **api/vcs** in the path.

For example, if you are using Artifactory standalone or as a local service, you would access your VCS repositories using the following URL:

```
http://localhost:8081/artifactory/api/vcs/<repository key>
```

Or, if you are using Artifactory SaaS, the URL would be:

```
https://<server name>.jfrog.io/<server name>/api/vcs/<repository key>
```

Artifactory exposes REST APIs that let you do the following with VCS repositories:

- [List all tags](#)
- [List all branches](#)
- [Download a specific tag](#)
- [Download a file within a tag](#)
- [Download a specific branch](#)
- [Download a file within a branch](#)
- [Download a release](#)

To help you build the API call correctly, you can select the VCS repository you want to interact with and click **Set Me Up**.

Tool

VCS

Repository

aql-elastic

General

Artifactory supports downloading tags or branches using a simple GET request. You can also specify to download a specific tag or branch as a tar.gz or zip, and a specific file within a tag or branch as a zip file.

Resolve

Use the following command to list all tags

```
1 curl -i -u<USERNAME>:<API_KEY> -XGET http://localhost:8080/artifactory/api/vcs/tags/aql-elastic  
/<USERR_ORG>/<REPO>
```

Use the following command to list all branches

```
1 curl -i -u<USERNAME>:<API_KEY> -XGET http://localhost:8080/artifactory/api/vcs/branches/aql-elastic  
/<USERR_ORG>/<REPO>
```

Get VCS Tags

Description: Lists all VCS tags.

Since: 3.6.0

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/vcs/tags/{repoKey}/{userOrg}/{repo}

Produces: application/json

Sample Output:

```
GET /api/vcs/tags/github/jquery/jquery  
[ {  
  "name" : "1.0",  
  "commitId" : "bcc8a837055fe720579628d758b7034d6b520f2e",  
  "isBranch" : false  
}, {  
  "name" : "1.0.1",  
  "commitId" : "bcc8a837055fe720579628d758b7034d6b520f2e",  
  "isBranch" : false  
}  
...]
```

Get VCS Branches

Description: Lists all VCS branches.

Since: 3.6.0

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/vcs/branches/{repoKey}/{userOrg}/{repo}

Produces: application/json

Sample Output:

```
GET /api/vcs/branches/github/jquery/jquery
[ {
  "name" : "1.11-stable",
  "commitId" : "852529c9f148de6df205be01659a79731ce8ebef",
  "isBranch" : true
}, {
  "name" : "1.x-master",
  "commitId" : "73c1ceaf4280bd0318679c1ad832181f3f449814",
  "isBranch" : true
}
...]
```

Download Tag

Description: Download a complete tarball (tar.gz/zip, default tar.gz) of a tag.

Downloading can be executed conditionally according to properties by specifying the properties query param. In this case only cached artifacts are searched.

Since: 3.6.0

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/vcs/downloadTag/{repoKey}/{userOrg}/{repo}/{tag-name}?ext=tar.gz/zip (default tar.gz)

Produces: application/octet-stream

Sample Output:

```
GET /api/vcs/downloadTag/github/jquery/jquery/2.0.1
<Tag binary content>
```

Download File within a Tag

Description: Download a specific file from within a tag.

Since: 3.6.0

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/vcs/downloadTagFile/{repoKey}/{userOrg}/{repo}/{tag-name}!{file-path}

Produces: application/octet-stream

Sample Output:

```
GET /api/vcs/downloadTagFile/github/jquery/jquery/2.0.1!AUTHORS.txt
<AUTHORS.txt content>
```

Download Branch

Description: Downloads a tarball (tar.gz/zip, default tar.gz) of a complete branch.

Downloading can be executed conditionally according to properties by specifying the properties query param. In this case only cached artifacts are searched.

Since: 3.6.0

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/vcs/downloadBranch/{repoKey}/{userOrg}/{repo}/{branch-name}?ext=tar.gz/zip[&properties=qa=approved]

Produces: application/octet-stream

Sample Output:

```
GET /api/vcs/downloadBranch/github/jquery/jquery/master
<Branch binary content>
```

Download File within a Branch

Description: Downloads a specific file from within a branch.

Since: 3.6.0

Security: Requires a privileged user (can be anonymous)

Usage: GET /api/vcs/downloadBranchFile/{repoKey}/{userOrg}/{repo}/{branch-name}!{file-path}

Produces: application/octet-stream

Sample Output:

```
GET /api/vcs/downloadBranchFile/github/jquery/jquery/master!README.md
<AUTHORS.txt content>
```

Download Release

Description: Downloads a complete release tarball (tar.gz/zip, default tar.gz) of a tag from GitHub.

Since: 4.3.0

Security: Requires a privileged user (can be anonymous)

VCS Usage: GitHub only

Usage: GET /api/vcs/downloadRelease/{repoKey}/{userOrg}/{repo}/{release-name}?ext=tar.gz/zip (default tar.gz)

Produces: application/octet-stream

Sample Output:

```
GET
/api/vcs/downloadRelease/git-remote/google/protobuf/v3.0.0-beta-1?ext=ta
r.gz/zip
<Tag binary content>
```

Examples

Below are some examples of working with the API using cURL:

Download jquery master branch from GitHub

```
curl -i
"http://localhost:8080/artifactory/api/vcs/downloadBranch/github/jquery/
jquery/master"
```

Download a specific tag from Bitbucket

```
curl -i
"http://localhost:8080/artifactory/api/vcs/downloadTag/bitbucket/lssystem
s/angular-extended-notifications/1.0.0"
```

Download a file within the tag 2.0.1 of jquery, '!' is escaped as '%21'

```
curl -i
"http://localhost:8080/artifactory/api/vcs/downloadTagFile/github/jquery/
jquery/2.0.1%21AUTHORS.txt"
```

When files are already cached, you can conditionally request them using a properties query param:

Download a file within the tag 2.0.1 of jquery, '!' is escaped as '%21'

```
curl -i
"http://localhost:8080/artifactory/api/vcs/downloadBranch/github/jquery/
jquery/2.0.1?properties=qa=approved"
```

Accessing Private VCS Repositories

Artifactory also supports accessing private VCS repositories such as a private GitHub or any self-hosted authenticated one.

To do so, simply add your credentials under **Advanced Settings** of the remote repository configuration panel.

Credentials when redirected

Some git providers (GitHub included) redirects download requests to a CDN provider.

You will need your credentials to pass along with the redirected request, simply check the **Lenient Host Authentication** and the credentials will pass transparently on each redirected request.

RPM Repositories

Overview

Artifactory is a fully-fledged RPM repository. As such, it enables:

1. RPM metadata calculation for RPMs hosted in Artifactory local repositories.
2. Provisioning RPMs directly from Artifactory to YUM clients.
3. Detailed RPM metadata views from Artifactory's web UI.
4. Providing GPG signatures that can be used by the YUM client to authenticate RPMs.

Valid for YUM also

The instructions on this page can be used for RPM repositories and YUM repositories interchangeably.

RPM Metadata for Hosted RPMs

The RPM metadata generated by Artifactory is identical to the basic-mode output of the Red Hat-based Linux command `createrepo`.

A folder named `repodata` is created in the configured location within a local repository with the following files in it:

File	Description
------	-------------

<i>primary.xml.gz</i>	Contains an XML file describing the primary metadata of each RPM archive.
<i>filelists.xml.gz</i>	Contains an XML file describing all the files contained within each RPM archive.
<i>other.xml.gz</i>	Contains an XML file describing miscellaneous information regarding each RPM archive.
<i>repomd.xml</i>	Contains information regarding all the other metadata files.

YUM Support is Platform Independent!

Artifactory's RPM metadata calculation is based on **pure Java**.

It does not rely on the existence of the `createrepo` binary or on running external processes on the host on which Artifactory is running.

Page Contents

- [Overview](#)
 - [RPM Metadata for Hosted RPMs](#)
 - [Triggering RPM Metadata Updates](#)
 - [Indexing the File List](#)
- [Configuration](#)
 - [Local Repositories](#)
 - [Remote Repositories](#)
 - [Virtual Repositories](#)
- [Signing RPM Metadata](#)
- [Using Yum to Install RPM Packages](#)
- [Using Yum to Deploy RPM Packages](#)
- [YUM Groups](#)
 - [Attaching a YUM Group](#)
 - [YUM Group Commands](#)
 - [Setting Group Properties](#)
- [Yum Authentication](#)
 - [Proxy Server Settings](#)
 - [SSL Setting](#)
- [Using Yum Variables](#)
- [Viewing Individual RPM Information](#)
 - [Metadata Fields as Properties](#)
- [Watch the Screencast](#)

Triggering RPM Metadata Updates

When enabled, the metadata calculation is triggered automatically by some actions, and can also be invoked manually by others. Either way, the metadata produced is served to YUM clients.

Automatic

RPM metadata is automatically calculated:

1. When deploying/removing/copying/moving an RPM file.
2. When performing content import (both system and repository imports).

Manual

You can manually invoke RPM metadata calculation:

1. By selecting the local repository in the Tree Browser and clicking **Recalculate Index** in the **Actions** menu.
2. Via Artifactory's [REST-API](#).

Metadata calculation cleans up RPM metadata that already existed as a result of manual deployment or import. This includes RPM metadata stored as SQLite database files.

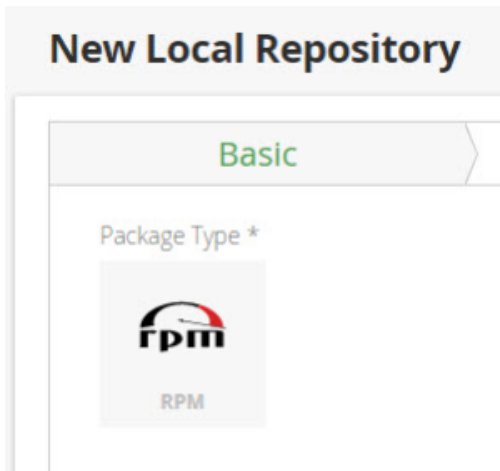
Indexing the File List

The `filelists.xml` metadata file of an RPM repository contains a list of all the files in each package hosted in the repository. When the repository contains many packages, reindexing this file as a result of interactions with the YUM client can be resource intensive causing a degradation of performance. Therefore, from version 5.4, reindexing this file is initially disabled when an RPM repository is created. To enable indexing `filelists.xml`, set the **Enable File List Indexing** checkbox.

Note that the `filelists.xml` metadata file for a virtual repository may not be complete (i.e. it may not actually list all the files it aggregates) if any of the repositories it aggregates do not have file listing enabled. Note that if indexing of the `filelists.xml` file is disabled, it is not possible to search for a file using the YUM client to determine which package wrote the queried file to the filesystem.

Configuration

To create an RPM local repository, select **RPM** as the **Package Type** when you create the repository.



Local Repositories

To enable automatic RPM metadata calculation on a local RPM repository, in the **RPM Settings** section of the **Basic** settings screen, set **Auto-calculate RPM Metadata**.

RPM Settings

RPM Metadata Folder Depth ?

Auto Calculate RPM Metadata

Enable File List Indexing

RPM Group File Names ?

 +

Field	Description
-------	-------------

<p>RPM Metadata Folder Depth</p>	<p>Informs Artifactory under which level of directory to search for RPMs and save the <code>repodata</code> directory.</p> <p>By default this value is 0 and refers to the repository's root folder. In this case, Artifactory searches the entire repository for RPM and saves the <code>repodata</code> directory at <code>\$REPO-KEY/repodata</code>.</p> <p>Using a different depth is useful in cases where generating metadata for a repository separates its artifacts by name, version or architecture. This will allow you to create multiple RPM repositories under the same Artifactory RPM repository.</p> <p>For example: If the repository layout is similar to that shown below and you want to generate RPM metadata for every artifact divided by name set the <code>Depth</code> to 1 and the <code>repodata</code> directory is saved at <code>REPO_ROOT/ARTIFACT_NAME/repodata</code> :</p> <div style="border: 1px dashed blue; padding: 10px; margin: 10px 0;"> <pre>REPO_ROOT/\$ARTIFACT_NAME/\$ARTIFACT_VERSION/\$ARCHITECTURE/FILE_NAME - or - rpm-local/foo/1.0/x64/foo-1.0-x64.rpm</pre> </div> <div style="border: 1px solid blue; padding: 10px; margin: 10px 0;"> <p>When changing the configured depth of existing repository, packages indexed in the old depth might need to be re-indexed or moved to a new depth to be available in the new configured depth, and YUM clients might need to change their configuration to point to the new <code>depth</code>.</p> </div>
<p>Auto-calculate RPM Metadata</p>	<p>When set, RPM metadata calculation is automatically triggered by the actions described above.</p>
<p>Enable File List Indexing</p>	<p>When set, RPM metadata calculation will also include indexing the <code>filelists.xml</code> metadata file.</p>
<p>RPM Group File Names</p>	<p>A comma-separated list of YUM group files associated with your RPM packages.</p> <p>Note that at each level (depth), the <code>repodata</code> directory in your repository may contain a different group file name, however each <code>repodata</code> directory may contain only 1 group metadata file (multiple groups should be listed as different tags inside the XML file). For more details, please refer to the YUM Documentation.</p>

Metadata calculation is asynchronous and does not happen immediately when triggered, whether [automatically](#) or [manually](#).

Artifactory invokes the actual calculation only after a certain "quiet period", so the creation of metadata normally occurs only 1-2 minutes after the calculation was triggered.

Remote Repositories

Artifactory remote repositories support RPMs out-of-the-box, and there is no need for any special configuration needed in order to work with RPMs in a remote repository.

All you need to do is point your YUM client at the remote repository, and you are ready to use YUM with Artifactory.


To define a remote repository to proxy an RPM remote repository, follow the steps below:

1. In the **Admin** module under **Repositories | Remote**, click "New" to create a new remote repository.
2. Set the **Repository Key** value, and specify the URL to the remote repository in the **URL** field as displayed below.

New Remote Repository

Basic > Advanced > Replications

Package Type *

 RPM

Repository Key *

URL *

3. Click "Save & Finish"
4. Back in the **Artifacts** module, in the **Tree Browser**, select the repository. Note that in the Tree Browser, the repository name is appended with "-cache".
5. Click **Set Me Up** and copy the value of the **baseurl** tag.

Set Me Up

Tool: RPM

Repository: targetCentos

Type password to insert your credentials to the code snippets

password

General

To resolve `.rpm` files using the YUM client, edit or create the `artifactory.repo` file with root privileges:

```
1 sudo vi /etc/yum.repos.d/artifactory.repo
```

Then edit the `baseurl` to point to the path of the `repopdata` folder according to configured repository depth. If the configured depth is 0 the `baseurl` should point to the root of the repository.

```
1 [Artifactory]
2 name=Artifactory
3 baseurl=http://<URL_ENCODED_USERNAME>:
  <PASSWORD>@localhost:8081/artifactory/targetCentos/<PATH_TO_REPODATA_FOLDER>
4 enabled=1
5 gpgcheck=0
6 #Optional - if you have GPG signing keys installed, use the below flags to verify the repository metadata
  signature:
7 #gpgkey=http://<URL_ENCODED_USERNAME>:
  <PASSWORD>@localhost:8081/artifactory/targetCentos/<PATH_TO_REPODATA_FOLDER>/repomd.xml.key
8 #repo_gpgcheck=1
```

6. Next, create the `/etc/yum.repos.d/targetCentos.repo` file and paste the following configuration into it:

```
[targetCentos]
name=targetCentos
baseurl=http://localhost:8081/artifactory/targetCentos/
enabled=1
gpgcheck=0
```

A Virtual Repository defined in Artifactory aggregates packages from both local and remote repositories.

This allows you to access both locally hosted RPM packages and remote proxied RPM repositories from a single URL defined for the virtual repository.

To define a virtual YUM repository, create a [virtual repository](#), set the **Package Type** to be **RPM**, and select the underlying local and remote RPM repositories to include in the **Basic** settings tab.

The screenshot shows the 'New Virtual Repository' configuration interface. The 'Basic' tab is selected. The 'Package Type' is set to 'RPM'. The 'Repository Key' is empty. Under 'General', 'Repository Layout' is set to 'simple-default'. There are fields for 'Public Description' and 'Internal Description'. Under 'Include Patterns', there is a 'New Pattern' field with a '+' icon. Under 'Exclude Patterns', there is a 'New Pattern' field with a '+' icon. Under 'RPM Settings', 'Metadata Retrieval Cache Period (Sec)' is set to '600'. On the right, the 'Repositories' section shows 'Available Repositories' with 'targetCentos' selected and 'Selected Repositories' which is empty. There are navigation arrows between the two lists. At the bottom right, there is a 'Default Deployment Repository' dropdown menu.

To allow deploying packages to this repository, set the [Default Deployment Repository](#).

Signing RPM Metadata

Artifactory supports using a GPG key to sign RPM metadata for authentication by the YUM client.

To generate a pair of GPG keys and upload them to Artifactory, please refer to [GPG Signing](#).

Using Yum to Install RPM Packages

After configuring the `rpm-local` repository in Artifactory, you need to configure your local machine to install software packages from it by executing the following steps:

1. Edit the `artifactory.repo` file with root privileges

```
sudo vi /etc/yum.repos.d/artifactory.repo
```

2. Paste the following configuration into the `artifactory.repo` file:

```
[Artifactory]
name=Artifactory
baseurl=http://localhost:8081/artifactory/rpm-local/
enabled=1
gpgcheck=0
```

Now, every RPM file deployed to the root of the `rpm-local` repository can be installed using:

```
yum install <package_name>
```

Using Yum to Deploy RPM Packages

Once you have configured your local machine to install RPM packages from your RPM local repository, you may also deploy RPM packages to the same repository [using the UI](#) or using the [REST API](#).

Through the REST API you also have the option to [deploy by checksum](#) or [deploying from an archive](#).

For example, to deploy an RPM package into a repository called *rpm-local* you could use the following:

```
curl -u<USERNAME>:<PASSWORD> -XPUT
http://localhost:8080/artifactory/rpm-local/<PATH_TO_METADATA_ROOT> -T
<TARGET_FILE_PATH>
```

where `PATH_TO_METADATA_ROOT` specifies the path from the repository root to the deploy folder.

YUM Groups

A YUM group is a set of RPM packages collected together for a specific purpose. For example, you might collect a set of "Development Tools" together as a YUM group.

A group is specified by adding a group XML file to same directory as the RPM packages included in it. The group file contains the metadata of the group including pointers to all the RPM files that make up the group.

Artifactory supports attaching a [YUM Group file](#) to the YUM calculation essentially mimicking the `createrepo -g` command.

A group file can also be created by running the following command:

```
sudo yum-groups-manager -n "My Group" --id=mygroup --save=mygroups.xml
--mandatory yum glibc rpm
```

Attaching a YUM Group

The process of attaching YUM group metadata to a local repository is simple:

1. Create an XML file in the groups format used by YUM. You can either just type it out manually using any text editor, or run the `yum-groups-manager` command from `yum-utils`.
2. Deploy the created group file to the `repodata` folder.
Artifactory will automatically perform the following steps:
 - Create the corresponding `.gz` file and deploy it next to the deployed group XML file.
 - Invoke a YUM calculation on the local repository.
 - Attach the group information (both the XML and the `.gz` file) to the `repomd.xml` file.
3. Make sure the group file names are listed in the **YUM Group File Names** field under the Basic tab of the repository configuration. This tells Artifactory which files should be attached as repository group information.

YUM Group Commands

The following table lists some useful YUM group commands:

Command	Description
<code>yum groupinstall <Group ID></code>	Install the YUM group. The group must be deployed to the root of the YUM local repository.
<code>yum groupremove <Group ID></code>	Remove the RPM group

<code>yum groupupdate <Group ID></code>	Update the RPM group. The group must be deployed to the root of the YUM local repository.
<code>yum groupinfo <Group ID></code>	List the RPM packages within the group.
<code>yum grouplist</code> more	List the YUM groups

Setting Group Properties

YUM group properties can be set in the `/etc/yum.config` file as follows:

Setting	Allowed values	Description
overwrite_groups	0 or 1	Determines YUM's behavior if two or more repositories offer package groups with the same name. If set to 1 then the group packages of the last matching repository will be used. If set to 0 then the groups from all matching repositories will be merged together as one large group.
groupremove_leaf_only	0 or 1	Determines YUM's behavior when the groupremove command is run. If set to 0 (default) then all packages in the group will be removed. If set to 1 then only those packages in the group that aren't required by another package will be removed.
enable_group_conditionals	0 or 1	Determines whether YUM will allow the use of conditionals packages. If set to 0 then conditionals are not allowed If set to 1 (default) package conditionals are allowed.
group_package_types	optional, default, mandatory	Tells YUM which type of packages in groups will be installed when <code>groupinstall</code> is called. Default is: default, mandatory

Yum Authentication

Proxy Server Settings

If your organization uses a proxy server as an intermediary for Internet access, specify the `proxy` settings in `/etc/yum.conf`. If the proxy server also requires authentication, you also need to specify the `proxy_username`, and `proxy_password` settings.

```
proxy=<proxy server url>
proxy_username=<user>
proxy_password=pass
```

If you use the yum plugin (`yum-rhn-plugin`) to access the ULN, specify the `enableProxy` and `httpProxy` settings in `/etc/sysconfig/rhn/up2date`. In addition, if the proxy server requires authentication, you also need to specify the `enableProxyAuth`, `proxyUser`, and `proxyPassword` settings as shown below.

```
enableProxy=1
httpProxy=<proxy server url>
enableProxyAuth=1
proxyUser=<user>
proxyPassword=<password>
```

SSL Setting

YUM supports SSL from version 3.2.27.

To secure a repository with SSL, execute the following steps:

- Generate a private key and certificate using [OpenSSL](#).
- Define your protected repository in a `.repo` file as follows:

```
[protected]
name = SSL protected repository
baseurl=<secure repo url>
enabled=1
gpgcheck=1
gpgkey=<URL to public key>
sslverify=1
sslclientcert=<path to .cert file>
sslclientkey=<path to .key file>
```

where:

gpgkey is a URL pointing to the ASCII-armored GPG key file for the repository . This option is used if YUM needs a public key to verify a package and the required key has not been imported into the RPM database.

If this option is set, YUM will automatically import the key from the specific URL. You will be prompted before the key is installed unless the **assumeyes** option is set.

Using Yum Variables

You can use and reference the following built-in variables in `yum` commands and in all YUM configuration files (i.e. `/etc/yum.conf` and all `.repo` files in the `/etc/yum.repos.d/` directory):

Variable	Description
<code>\$releasever</code>	This is replaced with the package's version, as listed in <code>distroverpkg</code> . This defaults to the version of the <code>redhat-release</code> package.
<code>\$arch</code>	This is replaced with your system's architecture, as listed by <code>os.uname()</code> in Python.
<code>\$basearch</code>	This is replaced with your base architecture. For example, if <code>\$arch=i686</code> then <code>\$basearch=i386</code>

The following code block is an example of how your `/etc/yum.conf` file might look:

```
[main]
cachedir=/var/cache/yum/$basearch/$releasever
keepcache=0
debuglevel=2
logfile=/var/log/yum.log
exactarch=1
obsoletes=1
gpgcheck=1
plugins=1
installonly_limit=3
[comments abridged]
```

Viewing Individual RPM Information

You can view all the metadata that annotates an RPM by choosing it in Artifactory's tree browser and selecting the **RPM Info** tab:

java-1.7.0-openjdk-1.7.0.60-2.4.3.0.fc20.x86_64.rpm Download Actions

General **Rpm Info** Effective Permissions Properties Watchers Builds Governance

Package Info

Name:	java-1.7.0-openjdk
Version:	1.7.0.60
Release:	2.4.3.0.fc20
Summary:	OpenJDK Runtime Environment
Epoch:	1
Build Date:	Thu Oct 17 17:11:43 UTC 2013
Size:	228516

Miscellaneous

URL:	http://openjdk.java.net/
Vendor:	Fedora Project
Packager:	Fedora Project
Build Host:	buildvm-14.phx2.fedoraproject.org
Source Rpm:	java-1.7.0-openjdk-1.7.0.60-2.4.3.0.fc20.src.rpm

Description

The OpenJDK runtime environment.

Provides

Name	Flags	Epoch	Version	Release	Pre
------	-------	-------	---------	---------	-----

< page 1 of 1 >

Metadata Fields as Properties

The corresponding RPM metadata fields are automatically added as properties of an RPM artifact in YUM repositories accessed through Artifactory:

- rpm.metadata.name
- rpm.metadata.arch
- rpm.metadata.version
- rpm.metadata.release
- rpm.metadata.epoch
- rpm.metadata.group
- rpm.metadata.vendor
- rpm.metadata.summary

Properties can be used for searching and other functions. For more details please refer to [Properties](#).

Watch the Screencast

Watch this short screencast to learn how easy it is to host RPMs in Artifactory.

Ecosystem Integration

Overview

As a universal artifact repository, Artifactory not only supports all major packaging formats, it is also integrated with all major build tools and CI servers currently available. In addition, Artifactory is tightly integrated with additional JFrog products such as [JFrog Bintray](#), [JFrog Mission Control](#) and [JFrog Xray](#).

Build Tool Plugins	Resolve artifacts through Artifactory and deploy build artifacts to repositories in Artifactory transparently with all common build tools like Maven , Gradle , Ivy and SBT .
CI System Plugins	Deploy your build artifacts into Artifactory directly from industry standard CI servers such as Jenkins , TeamCity , Bamboo and TFS/MSBuild .
JFrog Bintray	Integrate with JFrog Bintray Universal Distribution for a fully automated software delivery pipeline, end-to-end.
JFrog Mission Control	JFrog Mission Control provides universal repository management providing you with a centralized dashboard to manage all your enterprise Artifactory instance.
JFrog Xray	JFrog Xray provides universal artifact analysis for software artifacts, and reveals a variety of issues at any stage of the software application lifecycle.

Page Contents

- [Overview](#)

Read more

- [Maven Repository](#)
- [Working with Gradle](#)
- [Working with Ivy](#)

Maven Repository

Overview

As a Maven repository, Artifactory is both a source for artifacts needed for a build, and a target to deploy artifacts generated in the build process. Maven is configured using a `settings.xml` file located under your Maven home directory (typically, this will be `/user.home/.m2/settings.xml`). For more information on configuring Maven please refer to the [Apache Maven Project Settings Reference](#).

The default values in this file configure Maven to work with a default set of repositories used to resolve artifacts and a default set of plugins.

To work with Artifactory you need to configure Maven to perform the following two steps:

1. [Resolve artifacts through Artifactory](#)
2. [Deploy artifacts to repositories through Artifactory](#)

Once your Maven build is configured, Artifactory also provides tight integration with commonly used CI servers (such as [Jenkins](#), [TeamCity](#) or a [Bamboo](#)) through a set of plugins that you can freely install and use.

Page Contents

- [Overview](#)
- [Viewing Maven Artifacts](#)
- [Resolving Artifacts through Artifactory](#)
 - [Automatically Generating Settings](#)
 - [Provisioning Dynamic Settings for Users](#)
 - [Manually Overriding the Built-in Repositories](#)
 - [Additional Mirror Any Setup](#)
 - [Configuring Authentication](#)
- [Deploying Artifacts Through Artifactory](#)
 - [Setting Up Distribution Management](#)
 - [Setting Up Security in Maven Settings](#)
- [Watch the Screencast](#)

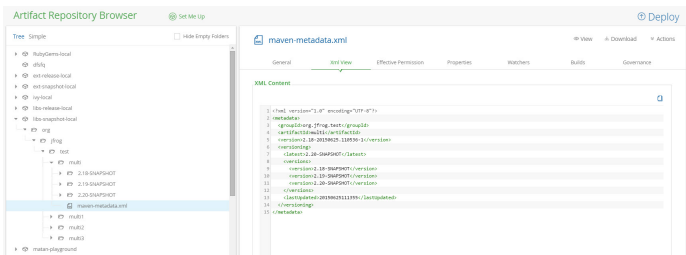
Read More

- [Maven Artifactory Plugin](#)

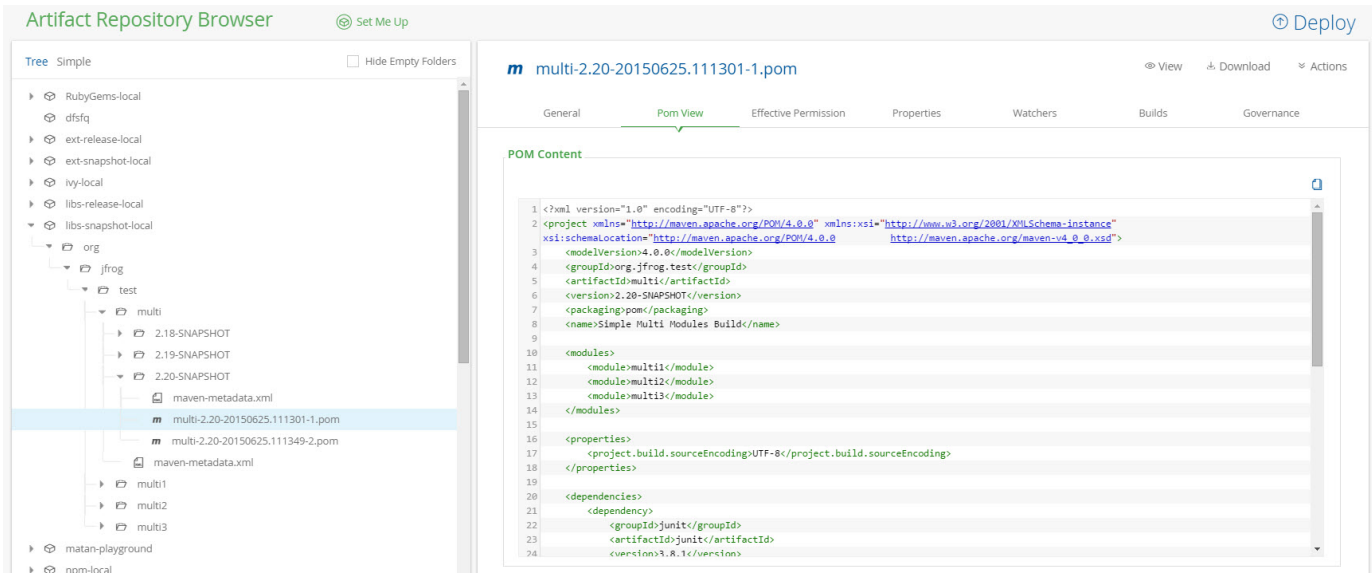
Viewing Maven Artifacts

If you select a Maven metadata file (maven-metadata.xml) or a POM file (pom.xml) in the Tree Browser, Artifactory provides corresponding tabs allowing you to view details on the selected item.

Maven Metadata View



POM View



Resolving Artifacts through Artifactory

To configure Maven to resolve artifacts through Artifactory you need to modify the `settings.xml`. You can generate one automatically, or modify it manually.

Automatically Generating Settings

To make it easy for you to configure Maven to work with Artifactory, Artifactory can automatically generate a `settings.xml` file which you can save under your Maven home directory.

The definitions in the generated `settings.xml` file override the default **central** and **snapshot** repositories of Maven.

In the **Artifact Repository Browser** of the **Artifacts** module, select **Set Me Up**. In the **Set Me Up** dialog, set **Maven** in the **Tool** field and click "Generate Maven Settings". You can now specify the repositories you want to configure for Maven.

Releases	The repository from which to resolve releases
Snapshots	The repository from which to resolve snapshots

Plugin Releases	The repository from which to resolve plugin releases
Plugin Snapshots	The repository from which to resolve plugin snapshots
Mirror Any	When set, you can select a repository that should mirror any other repository. For more details please refer to Additional Mirror Any Setup

Set Me Up
×

Tool

Maven

Back to Set Me Up

Releases ?

libs-release

Snapshots ?

libs-snapshot

Plugin Releases ?

plugins-release

Plugin Snapshots ?

plugins-snapshot

Mirror Any ?

remote-repos

Generate Settings

Once you have configured the settings for Maven you can click "Generate Settings" to generate and save the `settings.xml` file.

Provisioning Dynamic Settings for Users

You can deploy and provision a dynamic settings template for your users.

Once downloaded, settings are generated according to your own logic and can automatically include user authentication information.

For more details, please refer to the [Provisioning Build Tool Settings](#) under [Filtered Resources](#).

Manually Overriding the Built-in Repositories

To override the built-in **central** and **snapshot** repositories of Maven, you need to ensure that Artifactory is correctly configured so that no request is ever sent directly to them.

Using the automatically generated file as a template

You can use the automatically generated `settings.xml` file as an example when defining the repositories to use for resolving artifacts.

To do so, you need to insert the following into your parent POM or `settings.xml` (under an active profile):

```

<repositories>
  <repository>
    <id>central</id>
    <url>http://[host]:[port]/artifactory/libs-release</url>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
  </repository>
  <repository>
    <id>snapshots</id>
    <url>http://[host]:[port]/artifactory/libs-snapshot</url>
    <releases>
      <enabled>>false</enabled>
    </releases>
  </repository>
</repositories>
<pluginRepositories>
  <pluginRepository>
    <id>central</id>
    <url>http://[host]:[port]/artifactory/plugins-release</url>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
  </pluginRepository>
  <pluginRepository>
    <id>snapshots</id>
    <url>http://[host]:[port]/artifactory/plugins-snapshot</url>
    <releases>
      <enabled>>false</enabled>
    </releases>
  </pluginRepository>
</pluginRepositories>

```

Using the Default Global Repository

You can configure Maven to run with the [Default Global Repository](#) so that any request for an artifact will go through Artifactory which will search through all of the local and remote repositories defined in the system.

We recommend that you fine tune Artifactory to search through a more specific set of repositories by defining a dedicated virtual (or local) repository, and configure Maven to use that to resolve artifacts instead.

Additional Mirror Any Setup

In addition to [overriding built-in Maven repositories](#), you can use the **Mirror Any** setting to redirect all requests to a Maven repository through Artifactory, including those defined inside POMs of plug-ins and third party dependencies. (While it does not adhere to best practices, it is not uncommon for POMs to reference Maven repositories directly). This ensures no unexpected requests directly to Maven are introduced by such POMs.

You can either check **Mirror Any** in the **Maven Settings** screen when generating your *settings.xml* file, or you can manually insert the following:

```
<mirrors>
  <mirror>
    <id>artifactory</id>
    <mirrorOf>*</mirrorOf>
    <url>http://[host]:[port]/artifactory/[virtual repository]</url>
    <name>Artifactory</name>
  </mirror>
</mirrors>
```

Care when using "Mirror Any"

While this is a convenient way to ensure Maven only accesses repositories through Artifactory, it defines a coarse proxying rule that does not differentiate between releases and snapshots and relies on the single specified repository to do this resolution.

Using Mirrors

For more information on using mirrors please refer to [Using Mirrors for Repositories](#) in the Apache Maven documentation.

Configuring Authentication

Artifactory requires user authentication in three cases:

- Anonymous access has been disabled by unchecking the global **Allow Anonymous Access** setting.
- You want to restrict access to repositories to a limited set of users
- When deploying builds (while theoretically possible, it is uncommon to allow anonymous access to deployment repositories)

Authentication is configured in Maven using `<server>` elements in the `settings.xml` file.

Each `<repository>` and `<mirror>` element specified in the file must have a corresponding `<server>` element with a matching `<id>` that specifies the username and password.

The sample snippet below emphasizes that the `<repository>` element with `id=central` has a corresponding `<server>` element with `id=central`.

Similarly, the `<repository>` element with `id=snapshots` has a corresponding `<server>` element with `id=snapshots`.

The same would hold for `<mirror>` elements that require authentication.

In both cases the username is `admin` and the password is encrypted.

▼ Sample snippet from settings.xml

```
...
<servers>
  <server>
    <id>central</id>
    <username>admin</username>
    <password>{\DESede}\kFposSPUydYZf89Sy/o4wA==</password>
  </server>
  <server>
    <id>snapshots</id>
    <username>admin</username>
    <password>{\DESede}\kFposSPUydYZf89Sy/o4wA==</password>
  </server>
</servers>
<profiles>
  <profile>
```

```

    <repositories>
      <repository>
        <id>central</id>
        <snapshots>
          <enabled>>false</enabled>
        </snapshots>
        <name>libs-release</name>
        <url>http://localhost:8081/artifactory/libs-release</url>
      </repository>
    </repositories>
    <repository>
      <id>snapshots</id>
      <snapshots />
      <name>libs-snapshot</name>
      <url>http://localhost:8081/artifactory/libs-snapshot</url>
    </repository>
  </repositories>
</profile>
</profiles>
...

```

Artifactory encrypts passwords for safe and secure access to Maven repositories

To avoid having to use cleartext passwords, Artifactory [encrypts the password](#) in the settings.xml file that is generated. For example, in the above sample snippet we can see that the admin user name is specified in cleartext, but the password is encrypted:

```

<username>admin</username>
<password>\{DESede\}kFposSPUydYZf89Sy/o4wA==</password>

```

Synchronizing authentication details for repositories with the same URL

If you have repository definitions (either for deployment or download) that use *the same URL*, Maven takes the authentication details (from the corresponding server definition) of the first repository encountered and uses it for the life-time of the running build for all repositories with the same URL. This may cause authentication to fail (producing 401 errors for downloads or deployment) if you are using different authentication details for the respective repositories. This is inherent Maven behavior and can only be solved by using the same authentication details for all repository definitions with the same URL in your *settings.xml*.

Deploying Artifacts Through Artifactory

Setting Up Distribution Management

To deploy build artifacts through Artifactory you must add a deployment element with the URL of a target local repository to which you want to deploy your artifacts.

To make this easier, Artifactory displays a code snippet that you can use as your deployment element. In the **Artifacts** module **Tree Browser** select the repository you want to deploy to and click **Set Me UP**. The code snippet is displayed under **Deploy**.

Set Me Up



Tool

Maven

Generate Maven Settings

Repository

libs-release-local

General

Click on "Generate Maven Settings" in order to resolve artifacts through Virtual or Remote repositories.

Deploy

To deploy build artifacts through Artifactory you need to add a deployment element with the URL of a target local repository to which you want to deploy your artifacts. For example:

```
1 <distributionManagement>
2   <repository>
3     <id>Arti4-Demo</id>
4     <name>Arti4-Demo-releases</name>
5     <url>http://10.100.1.110:8081/artifactory/libs-release-local</url>
6   </repository>
7 </distributionManagement>
```

Remember that you can not deploy build artifacts to remote, so you should not use them in a deployment element.

Setting Up Security in Maven Settings

When deploying your Maven builds through Artifactory, you must ensure that any `<repository>` element in your distribution settings has a corresponding `<server>` element in the `settings.xml` file with a valid username and password as described in [Configuring Authentication](#) above. For the example displayed above, the Maven client expects to find a `<server>` element in the `settings.xml` with `<id>artifactory</id>` specified.

Anonymous access to distribution repository

If anonymous access to your distribution repository is allowed then there is no need to configure authentication. However, while it is technically possible, this is not good practice and is therefore an unlikely scenario

Watch the Screencast

Maven Artifactory Plugin

Overview

Artifactory supports Maven builds on commonly used build servers such as [Jenkins](#), [TeamCity](#) and [Bamboo](#) through corresponding plugins for these CI servers. However, in the last few years, the popularity of cloud-based build servers has grown spawning products like [Travis CI](#), [drone.io](#) and [Codeship](#).

The problem, is that none of these are "pluggable" in the traditional way. Therefore, to support Maven builds running on cloud-based build servers, you can use the **Maven Artifactory Plugin**.

You can use the Maven Artifactory plugin if a plugin for your CI server is not available (for example, cloud-based CI servers), or if you have very specific needs that are not supported by your CI server plugin.

Use only one plugin

However your build ecosystem is set up, make sure that you are only using one of the Artifactory plugins (either for your CI server, or for your build tool) to avoid clashing instructions and duplicated builds.

Source Code Available!

The Maven Artifactory Plugin is an [open-source project on GitHub](#) which you can freely browse and fork.

Page Contents

- [Overview](#)
- [Usage](#)
- [Configuration](#)
- [Reading Environment Variables and System Properties](#)

Through the Maven Artifactory Plugin, Artifactory is fully integrated with Maven builds and allows you to do the following:

1. Attach properties to published artifacts in Artifactory metadata.
2. Capture a [BuildInfo](#) object which can be passed to the [Artifactory REST API](#) to provide a fully traceable build context.
3. Automatically publish all build artifacts at the end of the build.

Usage

The Maven Artifactory Plugin coordinates are `org.jfrog.buildinfo:artifactory-maven-plugin:2.6.1`. It can be viewed on [Bintray](#), and can be download via the [JCenter Repository](#).

A typical build plugin configuration would be as follows:


```

<build>
  <plugins>
    ...
    <plugin>
      <groupId>org.jfrog.buildinfo</groupId>
      <artifactId>artifactory-maven-plugin</artifactId>
      <version>2.6.1</version>
      <inherited>>false</inherited>
      <executions>
        <execution>
          <id>build-info</id>
          <goals>
            <goal>publish</goal>
          </goals>
          <configuration>
            <deployProperties>
              <gradle>awesome</gradle>
              <review.team>qa</review.team>
            </deployProperties>
            <publisher>
              <contextUrl>https://oss.jfrog.org</contextUrl>
              <username>deployer</username>
              <password>{DESede}...</password>
              <repoKey>libs-release-local</repoKey>

            <snapshotRepoKey>libs-snapshot-local</snapshotRepoKey>
            </publisher>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

The plugin's invocation phase is "**validate**" by default and we recommend you don't change it so the plugin is called as early as possible in the lifecycle of your Maven build.

Configuration

The example above configures the Artifactory *publisher*, to deploy build artifacts either to the [releases](#) or the [snapshots](#) repository of the [public OSS instance of Artifactory](#) when `mvn deploy` is executed.

However, the Maven Artifactory Plugin provides many other configurations which you can see by running `mvn -X validate` and are displayed below:

```

<deployProperties> .. </deployProperties>
<artifactory>
  <envVarsExcludePatterns> .. </envVarsExcludePatterns>
  <envVarsIncludePatterns> .. </envVarsIncludePatterns>
  <includeEnvVars>true/false</includeEnvVars>
  <timeoutSec>N</timeoutSec>
</artifactory>

```

```
<publisher>
  <contextUrl> .. </contextUrl>
  <username> .. </username>
  <password> .. </password>
  <repoKey> .. </repoKey>
  <snapshotRepoKey> .. </snapshotRepoKey>
  <publishArtifacts>true/false</publishArtifacts>
  <publishBuildInfo>true/false</publishBuildInfo>
  <excludePatterns> .. </excludePatterns>
  <includePatterns> .. </includePatterns>

<filterExcludedArtifactsFromBuild>true/false</filterExcludedArtifactsFromBuild>
  <!-- If true build information published to Artifactory will include
  implicit project as well as build-time dependencies -->
  <recordAllDependencies>true/false</recordAllDependencies>
</publisher>
<buildInfo>
  <agentName> .. </agentName>
  <agentVersion> .. </agentVersion>
  <buildName> .. </buildName>
  <buildNumber> .. </buildNumber>
  <buildNumbersNotToDelete> .. </buildNumbersNotToDelete>
  <buildRetentionMaxDays>N</buildRetentionMaxDays>
  <buildRetentionCount>N</buildRetentionCount>
  <buildUrl> .. </buildUrl>
  <principal> .. </principal>
  <vcsRevision> .. </vcsRevision>
</buildInfo>
<licenses>
  <autoDiscover>true/false</autoDiscover>
  <includePublishedArtifacts>true/false</includePublishedArtifacts>
  <runChecks>true/false</runChecks>
  <scopes> .. </scopes>
  <violationRecipients> .. </violationRecipients>
</licenses>
<blackDuck>
  <appName> .. </appName>
  <appVersion> .. </appVersion>

<autoCreateMissingComponentRequests>true/false</autoCreateMissingComponentRequests>

<autoDiscardStaleComponentRequests>true/false</autoDiscardStaleComponentRequests>
  <includePublishedArtifacts>true/false</includePublishedArtifacts>
  <reportRecipients> .. </reportRecipients>
```

```
<scopes> .. </scopes>
</blackDuck>
```

<deployProperties>	Specifies properties you can attach to published artifacts. For example: <div style="border: 1px dashed blue; padding: 10px; margin: 10px 0;"> <pre><deployProperties> <groupId>\${project.groupId}</groupId> <artifactId>\${project.artifactId}</artifactId> <version>\${project.version}</version> </deployProperties></pre> </div>
<artifactory>	Specifies whether environment variables are published as part of <code>BuildInfo</code> metadata and which include or exclude patterns are applied when variables are collected
<publisher>	Defines an Artifactory repository where build artifacts should be published using a combination of a <code><contextUrl></code> and <code><repoKey>/<snapshotRepoKey></code> . Build artifacts are deployed if the <code>deploy</code> goal is executed and only after all modules are built
<buildInfo>	Updates <code>BuildInfo</code> metadata published together with build artifacts. You can configure whether or not <code>BuildInfo</code> metadata is published using the <code><publisher></code> configuration.
<licenses>	Controls auto-discovery and violation monitoring of third-party licenses
<blackDuck>	Configures Artifactory BlackDuck integration. Note that you need to specify <code><runChecks>>true</runChecks></code> to activate it.

Reading Environment Variables and System Properties

Every build server provides its own set of environment variables. You can utilize these variables when configuring the plugin as shown in the following example:

```
<publisher>

<contextUrl>{{ARTIFACTORY_CONTEXT_URL | "https://oss.jfrog.org"}}</contextUr
l>

...
</publisher>
<buildInfo>

<buildNumber>{{DRONE_BUILD_NUMBER | TRAVIS_BUILD_NUMBER | CI_BUILD_NUMBER | BUIL
D_NUMBER | "333"}}</buildNumber>
  <buildUrl>{{DRONE_BUILD_URL | CI_BUILD_URL | BUILD_URL}}</buildUrl>
</buildInfo>
```

Any plugin configuration value can contain several `{{ .. }}` expressions. Each expression can contain a single or multiple environment variables or system properties to be used.

The expression syntax allows you to provide enough variables to accommodate any build server requirements according to the following rules:

- Each expression can contain several variables, separated by a `'|'` character to be used with a configuration value
- The last value in a list is the default that will be used if none of the previous variables is available as an environment variable or a system property

For example, for the expression `{{v1|v2|"defaultValue"}}` the plugin will attempt to locate environment variable `v1`, then system property `v1`, then environment variable or system property `v2`, and if none of these is available, `"defaultValue"` will be used.

If the last value is not a string (as denoted by the quotation marks) and the variable cannot be resolved, `null` will be used (for example, for expression `{{v1|v2}}` where neither `v1` nor `v2` can be resolved).

You can attach additional artifacts to your module using the [Build Helper Maven Plugin](#).

Keeping your Artifactory publisher credentials secure

If you prefer to keep your Artifactory publisher credentials (username and password) secure (rather than providing them as free text in the plugin configuration), we recommend storing them as environment variables or system properties and have the plugin read them when needed. Since the usual Maven deploy does not support environment variables or system properties in `settings.xml`, this capability is unique to the Maven Artifactory Plugin.

Examples

The below project provides a working example of using the plugin:

- [Maven Artifactory Plugin](#)

Working with Gradle

Overview

Artifactory provides tight integration with Gradle. All that is needed is a simple modification of your `build.gradle` script file with a few configuration parameters.

Both the new and older publishing mechanisms of Gradle are supported, however some of the steps to configure the [Gradle Artifactory Plugin](#) depend on the version you are using, and these are detailed in the documentation pages.

The Gradle Artifactory Plugin can be used whether you are running builds using a CI server, or running standalone builds. In either case, you should note the following points:

1. CI Server Integration

When running Gradle builds in your continuous integration server, we recommend using one of the Artifactory Plugins for [Jenkins](#), [TeamCity](#) or [Bamboo](#).

You can use your build server UI to configure resolving and publishing artifacts through Artifactory to capture exhaustive build information.

2. Standalone Integration

The Gradle Artifactory plugin offers a simple DSL to perform the following steps in your Gradle build:

- Define the default dependency resolution from Artifactory.
- Define configurations that publish artifacts to Artifactory after a full (multi-module) successful build.
- Define properties that should be attached to published artifacts in Artifactory metadata.
- Capture and publish a [build-info](#) object to the Artifactory build-info REST API to provide a fully traceable build context.

Source Code Available!

This Gradle Artifactory Plugin is an [open source project on GitHub](#) which you can freely browse and fork.

The following sections describe the main configuration steps and provide a sample Gradle script that shows the information you need to get started using Gradle with Artifactory.

Page Contents

- [Overview](#)
- [Configuring Artifact Resolution](#)

- Using the Gradle Build Script Generator
- Provisioning Dynamic Settings for Users
- Sample Build Script and Properties
- Running Gradle
- Dependency Declaration Snippets
- Optimizing Gradle Builds
 - Configuring Artifactory
 - Configuring Gradle
 - Replication Across Different Sites
- Watch the Screencast

Read More

- [Gradle Artifactory Plugin](#)

Configuring Artifact Resolution

Using the Gradle Build Script Generator

With Artifactory's **Gradle Build Script Generator**, you can easily create a Gradle init script that handles resolution.

In the **Artifact Repository Browser** of the **Artifacts** module, select **Set Me Up**. In the **Set Me Up** dialog, set **Gradle** in the **Tool** field and click "Generate Gradle Settings". You can now specify the settings you want to configure for Gradle.

Plugin/Libs Resolver	The repository that should be used to resolve plugins/libraries
Use Maven/Use Ivy	When checked, specifies that resolving should be done using the Maven/Ivy pattern
Libs Publisher	The repository that should be used to publish libraries
Use Maven/Use Ivy	When checked, specifies that library should be published using a Maven/Ivy descriptor
Repository Layout	Specifies the layout of the corresponding repository

Once you have configured the settings for Gradle you can click "Generate Settings" to generate and save the `build.gradle` and `gradle.properties` file.

Set Me Up
✕

Tool

Gradle

Back to Set Me Up

Plugin Resolver

Repository Key ?

plugins-release

Use Maven ?

Use Ivy ?

Repository Layout ?

maven-2-default

Libs Resolver

Repository Key ?

libs-release

Use Maven ?

Use Ivy ?

Repository Layout ?

ivy-default

Libs Publisher

Repository Key ?

libs-release-local

Use Maven ?

Use Ivy ?

Repository Layout ?

maven-2-default

Generate Settings

Provisioning Dynamic Settings for Users

Artifactory lets you deploy and provision a dynamic settings template for your users. Once downloaded, settings are generated according to your own logic and can automatically include user authentication information.

For more details, please refer to [Provisioning Build Tool Settings](#) section under [Filtered Resources](#).

Sample Build Script and Properties

You can download sample scripts from the [JFrog GitHub public repository](#).

Running Gradle

For Gradle to build your project and upload generated artifacts to Artifactory, you need to run the following command:

```
gradle artifactoryPublish
```

For more details on building your projects with Gradle, please refer to the [Gradle Documentation](#).

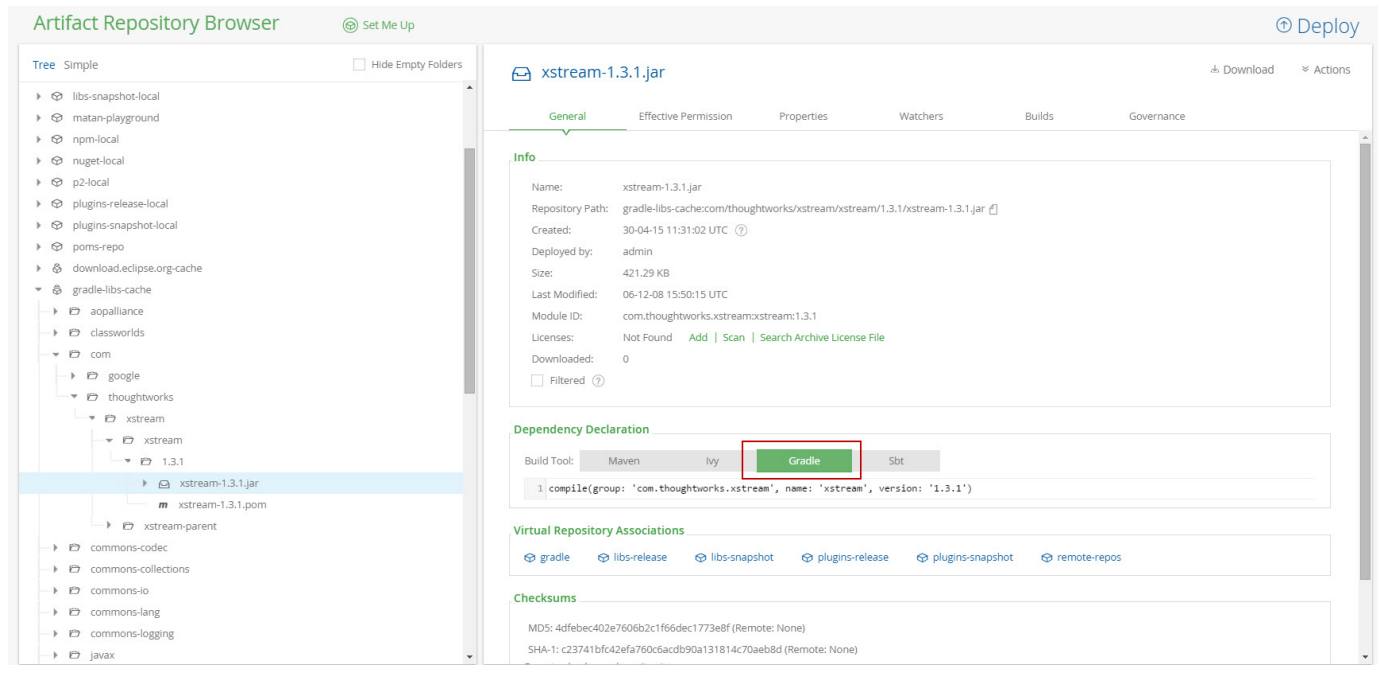
Getting debug information from Gradle

We highly recommend running Gradle with the `-d` option to get useful and readable information if something goes wrong with your build.

Dependency Declaration Snippets

Artifactory can provide you with dependency declaration code snippets that you can simply copy into the **Gradle Dependency Declaration** section of your `build.gradle` file.

In the **Artifact Repository Browser** of the **Artifacts** module, drill down in the repository tree and select a relevant artifact. Under the **Dependency Declaration** section, select **Gradle** to display the corresponding dependency declaration that you can copy into your `build.gradle` file.



Optimizing Gradle Builds

From V3.5, Gradle introduces a build cache feature that lets you reuse outputs produced by other builds, instead of rebuilding them, and dramatically reduce build time. This feature supports not only your local filesystem cache, but also remote caches that can be shared across your organization.

The Gradle team has measured an average reduction of 25% in total build time, and even a reduction of 80% with some of their commits!

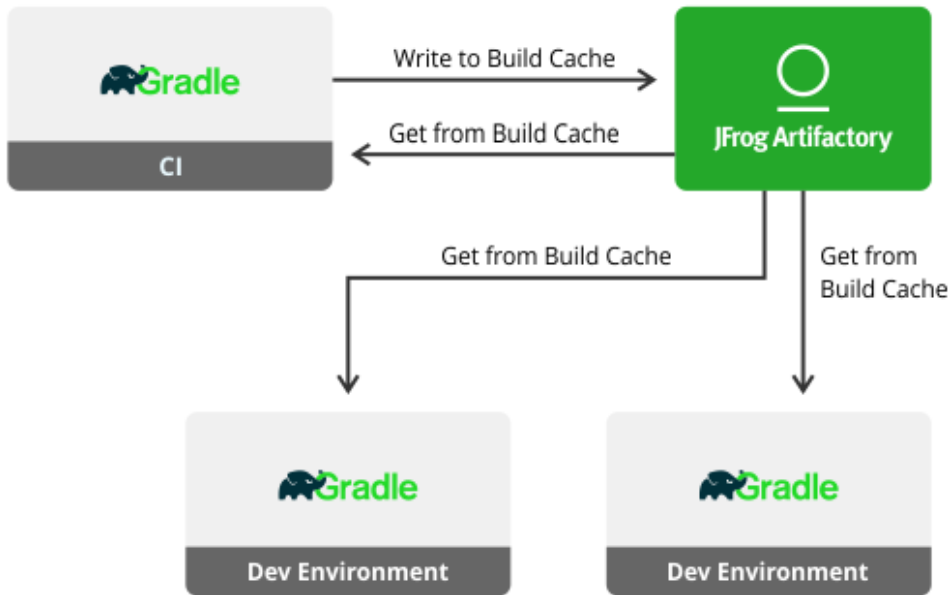
To optimize your Gradle builds:

1. [Configure Artifactory](#) to be your Gradle build cache
2. [Configure Gradle](#) to use the build cache in Artifactory

Configuring Artifactory

Artifactory can be used as the Gradle build cache by simply creating a [generic repository](#) in Artifactory.

For example, the following is a [simple use case](#) where the CI server builds a project and stores the build cache in Artifactory for later use by the following builds. This will greatly improve the build time in your local developer environments.



Configuring Gradle

Configure Gradle to use the build cache and point it to Artifactory.

gradle.properties

```

artifactory_user=admin
artifactory_password=password
artifactory_url=http://localhost:8081/artifactory
org.gradle.caching=true
gradle.cache.push=false
  
```

settings.gradle

Set the `gradle.cache.push` property to true, on the CI server, by overriding it using `-Pgradle.cache.push=true`.

```

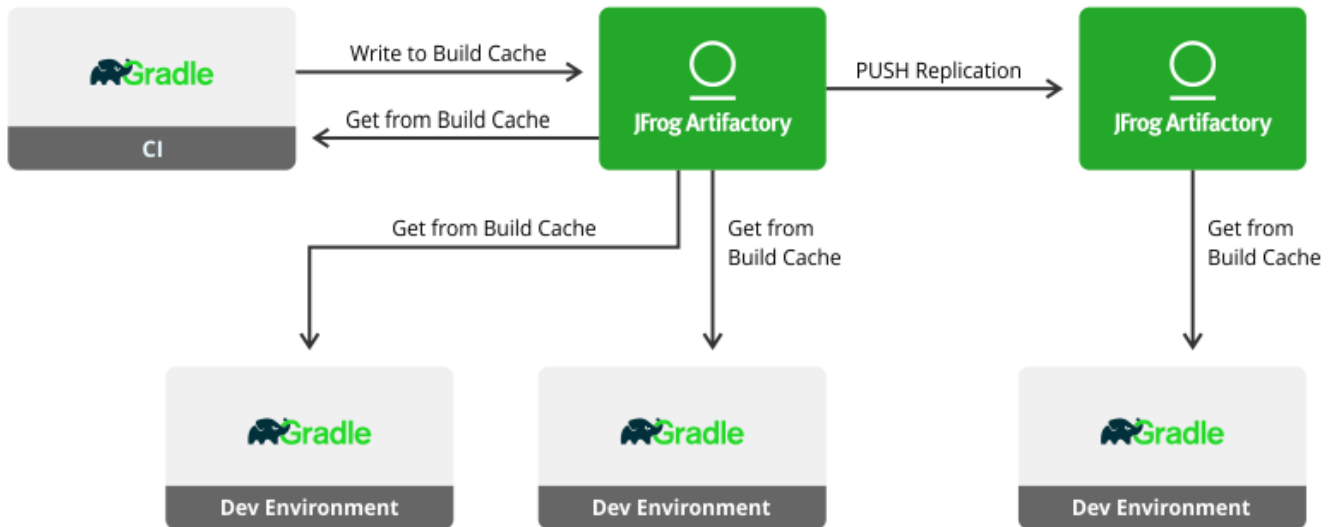
include "shared", "api", "services:webservice"

ext.isPush = getProperty('gradle.cache.push')

buildCache {
  local {
    enabled = false
  }
  remote(HttpBuildCache) {
    url = "${artifactory_url}/gradle-cache-example/"
    credentials {
      username = "${artifactory_user}"
      password = "${artifactory_password}"
    }
    push = isPush
  }
}
  
```


Replication Across Different Sites

You can also use Artifactory as a [distributed cache](#) that's synchronized across both local and remote teams using push and pull repository replication, and improve both your local and remote build times.



Watch the Screencast

Gradle Artifactory Plugin

Overview

The Gradle Artifactory Plugin allows you to deploy your build artifacts and build information to Artifactory and also to resolve your build dependencies from Artifactory.

Latest Version

For the latest version number of the Gradle Artifactory Plugin, please refer to the [download page on Bintray](#).

Download and Installation

Automatic Installation

Build script snippet for use in all Gradle versions

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath
        "org.jfrog.buildinfo:build-info-extractor-gradle:latest.release"
    }
}
apply plugin: "com.jfrog.artifactory"
```

Build script snippet for use in Gradle 2.1 and above

```
plugins {  
    id "com.jfrog.artifactory" version "latest.release"  
}
```

Currently the "plugins" notation cannot be used for applying the plugin for sub projects, when used from the root build script

Page Contents

- Overview
- Latest Version
- Download and Installation
 - Automatic Installation
- Manual Installation
- Configuration
 - Using the Artifactory Plugin DSL
 - The Artifactory Project Publish Task
 - Controlling Resolution and Publication in Sub-Projects
- Examples

Manual Installation

The latest plugin jar file can be [downloaded from JFrog Bintray](#). Download and copy the `build-info-extractor-gradle-<x.y.z>-uber.jar` into your gradle home plugins directory (`~/.gradle/plugins`).

Then add the following line to your project build script:

```
buildscript.dependencies.classpath files(new File(gradle.gradleUserHomeDir,  
'plugins/build-info-extractor-gradle-<x.y.z>-uber.jar'))
```

Configuration

Using the Artifactory Plugin DSL

The Gradle Artifactory plugin is configured using its own Convention DSL inside the `build.gradle` script of your root project.

The syntax of the Convention DSL is described below:

We highly recommend also using our [examples](#) as a reference when configuring the DSL in your build scripts.

Mandatory items within the relevant context are prefixed with '+'. All other items are optional.

```
artifactory {  
    +contextUrl = 'http://repo.myorg.com/artifactory' //The base  
    Artifactory URL if not overridden by the publisher/resolver  
    publish {  
        contextUrl = 'http://repo.myorg.com/artifactory' //The base  
        Artifactory URL for the publisher  
        //A closure defining publishing information  
        repository {
```

```

    +repoKey = 'integration-libs' //The Artifactory repository key to
publish to
    +username = 'deployer' //The publisher user name
    password = 'deployerPaS*' //The publisher password
    ivy {
        //Optional section for configuring Ivy publication. Assumes Maven
repo layout if not specified
        ivyLayout =
'[organization]/[module]/[revision]/[type]s/ivy-[revision].xml'
        artifactLayout =
'[organization]/[module]/[revision]/[module]-[revision](-[classifier]).[ex
t]'
        mavenCompatible = true //Convert any dots in an [organization]
layout value to path separators, similar to Maven's groupId-to-path
conversion. True if not specified
    }
}
defaults {
    //List of Gradle Publications (names or objects) from which to collect
the list of artifacts to be deployed to Artifactory.
    publications ('ivyJava','mavenJava','foo')
    ///List of Gradle Configurations (names or objects) from which to
collect the list of artifacts to be deployed to Artifactory.
    publishConfigs('archives', 'published')
    properties = ['qa.level': 'basic', 'q.os': 'win32, deb, osx']
//Optional map of properties to attach to all published artifacts
/*
    The properties closure in the "defaults" task uses the following
syntax:
    properties {
        publicationName 'group:module:version:classifier@type',
key1:'value1', key2:'value2', ...
    }
    publicationName: A valid name for a publication of the project. You
can use all to apply the properties to all publications.
    group:module:version:classifier@type: A filter that specifies the
artifacts to which properties should be attached.
    The filter may contain wildcards: * for all characters or ? for a single
character.
    key:'value': A list of key/value properties that will be attached
to to the published artifacts matching the filter.
*/
    properties {
//Optional closure to attach properties to artifacts based on a list of
artifact patterns per project publication
        foo '*:*:*:*@*', platform: 'linux', 'win64'
//The property platform=linux,win64 will be set on all artifacts in foo
publication
        mavenJava 'org.jfrog:*:*:*@*', key1: 'vall'
//The property key1=vall will be set on all artifacts part of the mavenJava
publication and with group org.jfrog
        all 'org.jfrog:shared:1.?:*~*', key2: 'val2', key3: 'val3'
//The properties key2 and key3 will be set on all published artifacts (all

```

```

publications) with group:artifact:version
    //equal to org.jfrog:shared:1.?
    }
    publishBuildInfo = true    //Publish build-info to Artifactory (true
by default)
    publishArtifacts = true    //Publish artifacts to Artifactory (true
by default)
    publishPom = true    //Publish generated POM files to Artifactory
(true by default).
    publishIvy = true    //Publish generated Ivy descriptor files to
Artifactory (true by default).
    }
}
resolve {
    contextUrl = 'http://repo.myorg.com/artifactory'    //The base
Artifactory URL for the resolver
    repository {
        +repoKey = 'libs-releases'    //The Artifactory (preferably virtual)
repository key to resolve from
        username = 'resolver'    //Optional resolver user name (leave out
to use anonymous resolution)
        password = 'resolverPaS*'    //The resolver password
        maven = true    //Resolve Maven-style artifacts and
descriptors (true by default)
        ivy {
            //Optional section for configuring Ivy-style resolution. Assumes
Maven repo layout if If not specified
            ivyLayout = ...
            artifactLayout = ...
            mavenCompatible = ...
        }
    }
}
// Redefine basic properties of the build info object
clientConfig.setIncludeEnvVars(true)
clientConfig.setEnvVarsExcludePatterns('*password*',*secret*')
clientConfig.setEnvVarsIncludePatterns('*not-secret*')
clientConfig.info.addEnvironmentProperty('test.adding.dynVar',new
java.util.Date().toString())
clientConfig.info.setBuildName('new-strange-name')
clientConfig.info.setBuildNumber('' + new
java.util.Random(System.currentTimeMillis()).nextInt(20000))
clientConfig.timeout = 600 // Artifactory connection timeout (in
seconds). The default timeout is 300 seconds.

```

```
}
```

Controlling how environment variables are exposed

As shown in the example above, you can control which environment variables are exposed in `clientConfig.setIncludeEnvVars` using `clientConfig.setEnvVarsExcludePatterns` and `clientConfig.setEnvVarsIncludePatterns`. These calls specify which environment variables should be excluded or included respectively using a parameter which is a comma-separated list of expressions to exclude or include. The expressions can use a star (*) wildcard to specify multiple environment variables.

Using the old Gradle publishing mechanism?

If you are using the old Gradle publishing mechanism, you need to replace the above defaults closure with the following one:

```
defaults {
    //This closure defines defaults for all 'artifactoryPublish' tasks
    of all projects the plugin is applied to
    publishConfigs ('a','b','foo')
    //Optional list of configurations (names or objects) to publish.

    //The 'archives' configuration is used if it exists and no configuration is
    specified
    mavenDescriptor = '/home/froggy/projects/proj-a/fly-1.0.pom'
    //Optional alternative path for a POM to be published (can be relative to
    project baseDir)
    ivyDescriptor = 'fly-1.0-ivy.xml'
    //Optional alternative path for an ivy file to be published (can be
    relative to project baseDir)
    properties = ['qa.level': 'basic', 'q.os': 'win32, deb, osx']
    //Optional map of properties to attach to all published artifacts
    /*
    The properties closure in the "defaults" task uses the following
    syntax:
    properties {
        configuration 'group:module:version:classifier@type',
    key1:'value1', key2:'value2', ...
    }
    configuration: A configuration that is a valid name of a
    configuration of the project. You can use all to apply the properties to
    all configurations.
    group:module:version:classifier@type: An artifact specification
    filter for matching the artifacts to which properties should be attached.
    The filter may contain wildcards: * for all characters or ? for a single
    character.
    key:'value': A list of key/value(s) properties that are attached to
    to the published artifacts matching the filter.
    */
    properties {
    //Optional closure to attach properties to artifacts based on a list of
    artifact patterns per project configuration
    foo '*:*:*:*@*', platform: 'linux', 'win64'
```

```
//The property platform=linux,win64 will be set on all artifacts in foo
configuration
    archives 'org.jfrog:*:*:*@*', key1: 'vall'
//The property key1=vall will be set on all artifacts part of the archives
configuration and with group org.jfrog
    all 'org.jfrog:shared:1.?:*@*', key2: 'val2', key3: 'val3'
//The properties key2 and key3 will be set on all published artifacts (all
configurations) with group:artifact:version

//equal to org.jfrog:shared:1.?
}
    publishBuildInfo = true    //Publish build-info to Artifactory (true
by default)
    publishArtifacts = true    //Publish artifacts to Artifactory (true
by default)
    publishPom = true          //Publish generated POM files to
Artifactory (true by default)
    publishIvy = false         //Publish generated Ivy descriptor files
to Artifactory (false by default)
```

```
}
```

The Artifactory Project Publish Task

The Artifactory Publishing Plugin creates an `artifactoryPublish` Gradle task for each project the plugin is applied to. The task is configured by the `publish` closure of the plugin.

You can configure the project-level task directly with the task's `artifactoryPublish` closure, which uses identical Syntax to that of the plugin's `publish.defaults` closure.

```
artifactoryPublish {
    skip = false //Skip build info analysis and publishing (false by
default)
    contextUrl = 'http://repo.myorg.com/artifactory'
    publications ('a','b','c')
    properties = ['qa.level': 'basic', 'q.os': 'win32, deb, osx']
    properties {
        c '**:**:*@*', cProperty: 'only in c'
    }
    clientConfig.publisher.repoKey = 'integration-libs'
    clientConfig.publisher.username = 'deployer'
    clientConfig.publisher.password = 'deployerPaS'
}
```

Controlling Resolution and Publication in Sub-Projects

The Gradle Artifactory Plugin allows you to define different resolution and publication configuration for sub projects. You may also define the configuration once for the whole project by defining the `artifactory` closure only in the root project. The plugin also lets you disable publication for a sub-module.

- When defining the configuration anywhere in the hierarchy, all sub-projects beneath it inherit the configuration and can override it whether it is defined in the root or in a sub-project.
- Each sub-project can override either the `publish` closure or the `resolve` closure, or both of them.

Example for overriding publication only

```
artifactory {
    publish {
        contextUrl = 'http://localhost:8081/artifactory'
        repository {
            repoKey = "libs-snapshot-local"
            username = "user"
            password = "pass"
        }
    }
}
```

Example for overriding resolution only

```
artifactory {
  resolve {
    contextUrl = 'http://localhost:8081/artifactory'
    repoKey = 'libs-snapshot'
    username = "user"
    password = "pass"
  }
}
```

Example for overriding both publication and resolution

```
artifactory {
  contextUrl = 'http://localhost:8081/artifactory'
  publish {
    repository {
      repoKey = "libs-snapshot-local"
      username = "admin"
      password = "password"
    }
  }
  resolve {
    repoKey = 'jcenter'
  }
}
```

- For buildInfo to be published, a publish closure must be defined in the root project.
- Use the `artifactoryPublish.skip` flag to deactivate analysis and publication.
- Activate the corresponding `artifactoryPublish` Gradle task manually for each project to which you wish to apply the plugin. For example in our [Gradle project example](#) you can run:

Activating the plugin manually

```
./gradlew clean api:artifactoryPublish shared:artifactoryPublish
```

Controlling the Build Name and Number

By default, BuildInfo is published with a build name constructed from the name of your root project and a build number that is the start date of the build.

You can control the build name and number values by specifying the following properties respectively:

Specifying the build name and number

```
buildInfo.build.name=my-super-cool-build
buildInfo.build.number=r9001
```

The above properties should be added to your project's `gradle.properties` file.

Examples

Project examples which use the Gradle Artifactory Plugin are available [here](#).

Working with Ivy

Overview

Artifactory fully supports working with Ivy both as a source for artifacts needed for a build, and as a target to deploy artifacts generated in the build process.

For Ivy to work with Artifactory, the following files must be present and configured:

1. **The Ivy settings file:** `ivysettings.xml` is used to configure resolution and deployment of artifacts using repositories in Artifactory.
2. **The Ivy modules file:** `ivy.xml` is where the project's modules and dependencies are declared.
3. **The Ant build file:** `build.xml` is used to execute the ANT tasks that will, in turn, use Ivy for resolution and deployment of artifacts.

Ivy Settings - `ivysettings.xml`

The `ivysettings.xml` file holds a chain of Ivy resolvers for both regular artifacts and Ivy module files. These are used to resolve and publish (i.e. deploy) artifacts.

There are a two ways to configure resolvers in `ivysettings.xml` in order to set up Ivy to work with Artifactory:

1. **Automatically**, using the Artifactory Ivy Settings Generator
2. **Manually** defining IBiblio and URL resolvers.

Page Contents

- Overview
- Ivy Settings - `ivysettings.xml`
 - Automatic Settings with Artifactory's Ivy Settings Generator
 - Provisioning Dynamic Settings for Users
 - Defining a Manual Resolver
 - The IBiblio Resolver
 - The URL Resolver
 - Using a Chain Resolver
- Ivy Modules - `ivy.xml`
- Ant Build - `build.xml`
- Publishing to Artifactory
 - Using a Dedicated Settings File for Deployment

Automatic Settings with Artifactory's Ivy Settings Generator

To begin quickly, you can define credentials and resolver settings using Artifactory's Ivy Settings Generator. This generates a URL resolver suitable for resolution.

In the **Artifact Repository Browser** of the **Artifacts** module, select **Set Me Up**. In the **Set Me Up** dialog, set **Ivy** in the **Tool** field and click "Generate Ivy Settings". You can now specify the repositories you want to configure for Ivy.

Since the **Libs Repository** field only includes virtual or remote repositories, none of these will be suitable for deployment, and you need to **modify the deployment URL** to point to a local repository.

Set Me Up ✕

Tool
Ivy ▼ [Back to Set Me Up](#)

Libs Repository ? Libs Repository Layout ? Libs Resolver Name ?
libs-release maven-2-default

Use Ibiblio Resolver ? Maven 2 Compatible ?

[Generate Settings](#)

Choose an Ivy Repository Layout

Be sure to select layout that is compatible with Ivy such as **ivy-default** or a custom layout that you have defined.

Provisioning Dynamic Settings for Users

You can deploy and provision a dynamic settings template for your users.

Once downloaded, settings are generated according to your own logic, and can automatically include user authentication information.

For details, please refer to [Provisioning Build Tool Settings](#) under [Filtered Resources](#).

Defining a Manual Resolver

The IBiblio Resolver

This resolver is only used to resolve dependencies. By default, it assumes artifacts in your repository are laid-out in the popular and standard Maven 2 format (which may not always be the case).

The [IBiblio resolver](#) can resolve artifacts from remote Maven 2 HTTP repositories, and if you use version ranges it relies on `maven-metadata.xml` files in the remote repository to gather information on the available versions.

To use the IBiblio resolver, add the following to your `ivysettings.xml` file:

```
<resolvers>
  <ibiblio name="artifactory" m2compatible="true"
  root="http://localhost:8080/artifactory/libs-releases"/>
</resolvers>
```

The URL specified in the `root` property must point to an Artifactory repository. In the above example, it is the pre-configured `libs-releases` virtual repository.

The `m2compatible` property configures the resolver with an artifact pattern that follows the standard Maven 2 layout.

The URL Resolver

The URL resolver can be used to resolve dependencies and/or for deployment of both regular artifacts and Ivy module files.

To publish or resolve artifacts to or from Artifactory, you need to configure a URL resolver with the pattern that matches your target repository layout for both Ivy and artifact files.

For example:

```

<!-- Authentication required for publishing (deployment). 'Artifactory
Realm' is the realm used by Artifactory so don't change it. -->
<credentials host="localhost" realm="Artifactory Realm" username="admin"
passwd="password"/>
<resolvers>
  <url name="artifactory-publish">
    <!-- You can use m2compatible="true" instead of specifying your
own pattern -->
    <artifact pattern=

"http://localhost:8080/artifactory/ivy-local/[organization]/[module]/[revi
sion]/[artifact]-[revision].[ext]"/>
    <ivy
pattern="http://localhost:8080/artifactory/ivy-local/[organization]/[modul
e]/[revision]/ivy-[revision].xml" />
    </url>
  </resolvers>

```

The URL resolver uses HTML href analysis to learn about the available versions of a remote artifact. This is less reliable than using an IBiblio resolver, however it works well with remote Artifactory servers.

Using a Chain Resolver

You can combine resolver definitions under a chain resolver in Ivy which uses a set of sub resolvers to resolve dependencies and for publishing.

For details please refer to the Ivy documentation for [Chain Resolver](#).

Ivy Modules - ivy.xml

ivy.xml files contain a list of dependency declarations that must be resolved for the build.

In the **Artifact Repository Browser** of the **Artifacts** module, you can obtain dependency declaration snippets by selecting either an Ivy module, or a POM artifact, and copying the Ivy **Dependency Declaration** section into your ivy.xml file.

The screenshot shows the 'Artifact Repository Browser' interface. On the left, a tree view shows the hierarchy of artifacts, with 'ivy-1.0-local-20120928001043.xml' selected. The main panel displays the details for this artifact, including its name, repository path, creation date, and size. Below the 'Info' section, the 'Dependency Declaration' section is visible, showing a snippet of XML code for an Ivy dependency declaration. The 'Ivy' build tool option is highlighted in the 'Build Tool' dropdown menu.

Info

Name: ivy-1.0-local-20120928001043.xml
Repository Path: ivy-local:org.apache.ivy.example/verston/1.0-local-20120928001043/ivys/ivy-1.0-local-20120928001043.xml
Created: 30-04-15 11:30:59 UTC
Deployed by: admin
Size: 1004 bytes
Last Modified: 27-09-12 22:10:56 UTC
Module ID: org.apache.ivy.example:verston:1.0-local-20120928001043:ivy
Licenses: Not Found | Add | Scan
Downloaded: 0
 Filtered

Dependency Declaration

Build Tool: Maven Ivy Gradle Sbt

```

1 <dependency org="org.apache.ivy.example" name="version" rev="1.0-local-20120928001043">
2   <artifact name="version" type="ivy" ext="xml"/>
3 </dependency>

```

Ant Build - build.xml

To work with Ivy to resolve dependencies, you need to use `<ivy:configure/>` in your `build.xml` file. This will load the Ivy settings from `ivysettings.xml`.

Artifacts are resolved using `<ivy:retrieve/>`.

For details please refer to the Ivy documentation for [Ant Tasks](#).

Publishing to Artifactory

You can use the `<ivy:publish>` command to configure Ivy to deploy your artifacts into Artifactory using the specified resolver.

For example:

```
<ivy:publish resolver="artifactory-publish" overwrite="true">
  <!--
    Use overwrite="true" if you wish to overwrite existing artifacts
    and publishivy="false" if you only want to publish artifacts not module
    descriptors
  -->
  <artifacts/>
</ivy:publish>
```

Using a Dedicated Settings File for Deployment

If you have specified deployment settings with the required credentials in a dedicated settings file, you can refer to them by assigning a unique ID.

For example, the following code snippet assigns the deployment settings with the id `ivy.publish.settings`:

```
<ivy:settings id="ivy.pub.settings"
file="publish_to_artifactory_settings.xml" />
```

Then, the publishing task points to these settings using the following attribute in the `publish` element:

```
settingsRef="ivy.pub.settings"
```

For details please refer to the Ivy documentation for [Ant Tasks](#).

Build Integration

Overview

Artifactory supports build integration whether you are running builds on one of the common CI servers in use today, on cloud-based CI servers or standalone without a CI server.

Integration of Artifactory into your build ecosystem provides important information that supports fully reproducible builds through visibility of artifacts deployed, dependencies and information on the build environment.

The Artifactory Build Integration Add-on provides a set of plugins you can use with industry standard CI systems and build tools that enable you to:

- See all the builds that are published and their build results in Artifactory.
- Explore the modules of each build, including published artifacts and corresponding dependencies.
- Obtain information about the build environment.
- Check if a specific artifact is required for or is a result of a build, and providing alerts if such an artifact should be targeted for removal.
- Treat all the artifacts and/or dependencies from a specific build as a single unit and perform bulk

- operations such as move, copy, export etc.
- Receive bidirectional links between build and artifact information inside the build server and Artifactory pages.

Running Builds on a CI Server

Artifactory can easily be added to a continuous integration build ecosystem by treating the CI server as a regular build client, so that it resolves dependencies from Artifactory, and deploys artifacts into a dedicated repository within Artifactory.

Supported Plugins

CI servers that are currently supported, each through a specific plugin are:

- **Jenkins/Hudson**
- **TeamCity**
- **Bamboo**
- **TFS**

The build tools supported on all of these CI servers are: **Maven 3 and 2**, **Gradle**, **Ivy/Ant**, **.Net**, **MSBuild** as well as **Generic** build tools. For details please refer to the documentation for each CI server plugin.

Bintray Plugins

In addition to the plugins supported by Artifactory, there is also the [Gradle Bintray Plugin](#) which provides integration between Gradle builds directly with JFrog Bintray.

Running Standalone Builds or on a Cloud-based CI Server

In the last few years, the popularity of cloud-based CI servers has grown. Some examples are, [CircleCI](#), [Travis CI](#), [drone.io](#) and [Codeship](#). The problem is that none of these are "pluggable" in the traditional way.

Therefore, to support builds running on cloud-based build servers, as well as standalone builds, Artifactory provides plugins for industry standard build tools such as Maven, Gradle, Ivy/Ant and MSBuild. These plugins provide all the benefits of Artifactory that facilitate fully reproducible builds without the need for a CI server. For more details please refer to [Maven Repository](#), [Working with Gradle](#), [Working with Ivy](#) and [MSBuild Artifactory Plugin](#).

Build Integration for Artifactory open source version vs. Artifactory Pro

When using the OSS version of Artifactory, Build Integration includes the Generic `BuildInfoView` and the ability to traverse and view build information using Artifactory's REST APIs.

[Artifactory Power Pack](#) extends these capabilities and provides Module Artifacts and Dependencies View, Repository View of Builds and the ability to export and manipulate build items.

Page Contents

- [Overview](#)
 - [Running Builds on a CI Server](#)
 - [Supported Plugins](#)
 - [Running Standalone Builds or on a Cloud-based CI Server](#)
- [Inspecting Builds](#)
 - [Builds and Build History](#)
 - [Build-level Information](#)
 - [General Build Information](#)
 - [Published Modules](#)
 - [Module Artifacts and Dependencies](#)
 - [Environment](#)
 - [Issues](#)
 - [Licenses](#)
 - [Build Diff](#)
 - [Release History](#)
 - [Build Info JSON](#)
 - [Generic BuildInfo View](#)
- [Exporting and Manipulating Build Items](#)
- [Repository View of Builds](#)
- [Behind the Scenes](#)
- [Release Management](#)

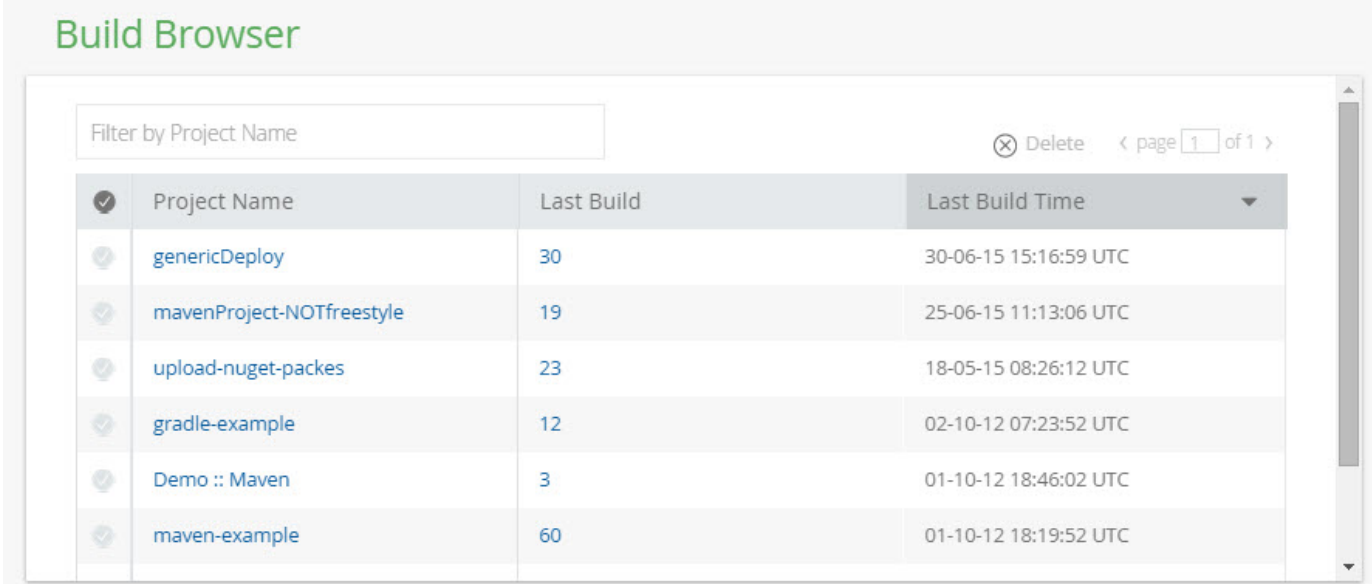
Read More

- [Jenkins Artifactory Plug-in](#)
- [TeamCity Artifactory Plug-in](#)
- [Bamboo Artifactory Plug-in](#)
- [MSBuild Artifactory Plugin](#)
- [VS Team Services Artifactory Plugin](#)
- [Using File Specs](#)

Inspecting Builds

Builds and Build History

All CI server projects that deploy their output to Artifactory can be viewed in the **Build Browser** which is accessed in the **Artifacts** module under **Builds**.



The screenshot shows the 'Build Browser' interface. At the top, there is a search bar labeled 'Filter by Project Name'. To the right, there are controls for 'Delete' (with a trash icon) and 'page 1 of 1'. Below these is a table with the following columns: 'Project Name', 'Last Build', and 'Last Build Time'. The table contains six rows of data:

<input checked="" type="checkbox"/>	Project Name	Last Build	Last Build Time
<input checked="" type="checkbox"/>	genericDeploy	30	30-06-15 15:16:59 UTC
<input checked="" type="checkbox"/>	mavenProject-NOTfreestyle	19	25-06-15 11:13:06 UTC
<input checked="" type="checkbox"/>	upload-nuget-packes	23	18-05-15 08:26:12 UTC
<input checked="" type="checkbox"/>	gradle-example	12	02-10-12 07:23:52 UTC
<input checked="" type="checkbox"/>	Demo :: Maven	3	01-10-12 18:46:02 UTC
<input checked="" type="checkbox"/>	maven-example	60	01-10-12 18:19:52 UTC

Selecting a project displays all runs of that build reflecting the build history in the CI server.

Build Browser

History for Build 'maven-example'

Filter by Build Number

Delete < page 1 of 1 >

✓	Build Number	CI Server	Number Of Modules/Artif...	Status	Last Build Time
✓	60	http://repo-demo:8080/jenkin...	Modules-4, Artifacts-9, Depen...	Staged	01-10-12 18:19:52 UTC
✓	59	http://repo-demo:8080/jenkin...	Modules-4, Artifacts-9, Depen...		01-10-12 16:17:51 UTC
✓	58	http://repo-demo:8080/jenkin...	Modules-4, Artifacts-9, Depen...		27-09-12 22:57:26 UTC
✓	57	http://repo-demo:8080/jenkin...	Modules-4, Artifacts-9, Depen...	Released	27-09-12 22:55:22 UTC
✓	56	http://repo-demo:8080/jenkin...	Modules-4, Artifacts-9, Depen...	Staged	27-09-12 22:52:49 UTC
✓	55	http://repo-demo:8080/jenkin...	Modules-4, Artifacts-9, Depen...		27-09-12 22:49:23 UTC
✓	53	http://repo-demo:8080/jenkin...	Modules-4, Artifacts-9, Depen...	Staged	27-09-12 22:35:56 UTC
✓	52	http://repo-demo:8080/jenkin...	Modules-4, Artifacts-9, Depen...		27-09-12 22:33:12 UTC
✓	51	http://repo-demo:8080/jenkin...	Modules-4, Artifacts-9, Depen...		27-09-12 22:26:57 UTC

Selecting a build item from the list displays complete **build-level information**. You can also view the build in the CI server by selecting the corresponding link under the **CI Server** column.

Permissions

To view build information you must have the 'deploy' permission on some repository path.

Build-level Information

You can select the **Build Number** to drill down into a specific build. This displays detailed information about the build, and enables you to compare it with another build as described in the following sections.

There are three categories of information:

1. **General build information** about the build and its environment.
2. **Build modules** along with their **artifacts and dependencies**.
3. Generic view of the build information in JSON format.

General Build Information

This tab displays general information about the build:


Build Browser

Build #767

General Build info | Published Modules | Environment | Issues | Licenses | Governance | Diff | Release History | Build info JSON

General Info

Name:	maven-project
Number:	767
Agent:	Jenkins/1.607
Build Agent:	Maven/3.2.5
Started:	2015-07-17T21:02:36.992+0000
Duration:	6.9 seconds
Artifactory Principal:	admin
URL:	http://dima:8080/job/maven-project/767/



Distribute

Name	The name assigned to the component being built
Number	The specific run of the build
Type	The build tool used
Agent	The CI server managing the build
Build Agent	The specific version of build tool used
Started	The time stamp when the build was started
Duration	The duration of the build
Principal	The factor that triggered this build. This may be a CI server user, or another build
Artifactory Principal	The Artifactory user that triggered this build
URL	Link to the build information directly on the build server

Published Modules

This tab displays the modules published into Artifactory as a result of the build, along with the number of artifacts and dependencies that they contain.

Build Browser

Build #60

General Build Info **Published Modules** Environment Issues Licenses Governance Diff >>

4 matches found

Filter by Module ID < page 1 of 1 >

Module ID	Number Of Artifacts	Number Of Dependencies
org.jfrog.test:multi1:2.1.8	4	13
org.jfrog.test:multi2:2.1.8	2	1
org.jfrog.test:multi3:2.1.8	2	15
org.jfrog.test:multi:2.1.8	1	0

Module Artifacts and Dependencies

Selecting a published module that was built will display its artifacts and dependencies. You can group these by type or scope by clicking the corresponding column header.

You can click any item to download it directly, or click its **Repo Path** to view it in the [Tree Browser](#).

All Builds > maven-example > 60

Build Browser

Build #60

General Build Info | **Published Modules** | Environment | Issues | Licenses | Governance | Diff | **Release History** | Build Info JSON

4 matches found
Filter by Module ID < page 1 of 1 >

Module ID	Number Of Artifacts	Number Of Dependencies
org.jfrog.test:multi1:2.1.8	4	13
org.jfrog.test:multi2:2.1.8	2	1
org.jfrog.test:multi3:2.1.8	2	15
org.jfrog.test:multi:2.1.8	1	0

Module Details: org.jfrog.test:multi2:2.1.8

Compare with previous build

Artifacts

2 matches found
Filter by Artifact Name < page 1 of 1 >

Artifact Name	Type	Repo Path
multi2-2.1.8.jar	jar	
multi2-2.1.8.pom	pom	

Dependencies

1 matches found
Filter by Dependency ID < page 1 of 1 >

Dependency ID	Scope	Type	Repo Path
junit:junit:3.8.1	test	jar	gradle-libs-cache/junit/junit/3.8.1/junit-3.8.1.jar

Environment

The Environment tab displays an extensive list of properties and environment settings defined for the selected build. You can use these to reproduce the environment precisely if you need to rerun the build.

All Builds > maven-example > 60

Build Browser

Build #60

General Build Info | Published Modules | **Environment** | Issues | Licenses | Governance | Diff | >>

Environment Variables

< page 1 of 3 >

Key	Value
buildInfo.env._	/System/Library/Frameworks/JavaVM.framework/Versions/Current/DK/Home/bin/java
buildInfo.env._CF_USER_TEXT_ENCODING	0x1F5:0:0
buildInfo.env.Apple_PubSub_Socket_Render	/tmp/launch-c42X4o/Render
buildInfo.env.Apple_Ubiquity_Message	/tmp/launch-Wgt166/Apple_Ubiquity_Message
buildInfo.env.BUILD_CAUSE	USERIDCAUSE
buildInfo.env.BUILD_CAUSE_USERIDCAUSE	true
buildInfo.env.BUILD_ID	2012-10-01_20-18-46
buildInfo.env.BUILD_NUMBER	60

Issues

The **Issues** provides integration between Artifactory, Jenkins CI server and JIRA issue tracker. When using Jenkins CI, if you to set the [Enable JIRA Integration](#) option in the Jenkins Artifactory Plugin, the **Issues** tab will display any JIRA issues that have been addressed by this build.

All Builds > maven-example > 60

Build Browser

Build #60

General Build Info | Published Modules | Environment | **Issues** | Licenses | Governance | Diff >>

Key	Summary	Previous Build
RTFACT-4931	Saving large artifact.config.xml from the UI throw internal e...	
RTFACT-4932	Download latest version/integration for non-Maven artifacts	

Licenses

The **Licenses** tab displays the results of a detailed license analysis of all artifacts and their dependencies.

All Builds > maven-example > 60

Build Browser

Build #60

General Build Info | Published Modules | Environment | Issues | **Licenses** | Governance | Diff >>

Summary: Unapproved: 3 Not Found: 13 Unknown: 0 Neutral: 0 Approved: 2

Includes

Include Published Artifacts

Include dependencies of the following scopes:

compile test provided runtime

[Export to CSV](#) [Auto-find Licenses](#)

Filter by Artifact ID < page 1 of 1 >

Artifact ID	Scopes	Repo Path	License
aopalliance:aopalliance:1.0	compile	gradle-libs-cache:aopalliance/aopalliance/1.0...	Public Domain
org.springframework:spring-beans:2.5.6	compile	Not in repository (externally resolved or dele...	Not Found
org.springframework:spring-aop:2.5.6	compile	Not in repository (externally resolved or dele...	Not Found
org.springframework:spring-core:2.5.6	compile	Not in repository (externally resolved or dele...	Not Found
org.testng:testng:5.9	test	gradle-libs-cache:org/testng/testng/5.9/testn...	Not Found
javax.servlet:servlet-api:2.5	provided	java.net.m1-cache:javax/servlet/servlet-api/2...	Not Found
javax.mail:mail:1.4	compile	gradle-libs-cache:javax/mail/mail/1.4/mail-1...	Not Found

The **Summary** line displays the number of artifacts found with the following statuses:

Unapproved	<p>The license found has not been approved for use</p> <div style="border: 1px solid green; padding: 10px; margin-top: 10px;"> <p>Approving licenses</p> <p>You can approve a license for use in the Admin tab under Configuration Licenses. For details please refer to License Control.</p> </div>
-------------------	---

Not Found	No license requirements were found for the artifact.
Unknown	The artifact requires a license that is unknown to Artifactory
Neutral	A license requirement that is not approved has been found for the artifact, however there is another license that is approved.
Approved	All license requirements for the artifact are approved in Artifactory.

Build Diff

The **Diff** tab allows you to compare the selected build with any other build. Once you select a build number in the **Select A Build To Compare Against** field, Artifactory displays all the differences between the builds that were detected including new artifacts added, dependencies deleted, properties changed and more.

All Builds > maven-example > 60

Build Browser

Build #60

General Build Info | Published Modules | Environment | Issues | Licenses | Governance | **Diff** >>

Select A Build To Compare Against:

59

Exclude Internal Dependencies

Artifacts (9 Results)

Name (Current Build)	Name (Build #59)	Status	Module	Repo Path
multi2-2.1.8.jar	multi2-2.1.8.jar	Updated	org.jfrog.test:multi2:2.1.8	
multi3-2.1.8.war	multi3-2.1.8.war	Updated	org.jfrog.test:multi3:2.1.8	
multi-2.1.8.pom	multi-2.1.8.pom	Updated	org.jfrog.test:multi:2.1.8	http://10.100.1.110:8081/artifacto...
multi2-2.1.8.pom	multi2-2.1.8.pom	Updated	org.jfrog.test:multi2:2.1.8	http://10.100.1.110:8081/artifacto...
multi1-2.1.8-tests.jar	multi1-2.1.8-tests.jar	Updated	org.jfrog.test:multi1:2.1.8	
multi1-2.1.8-sources.jar	multi1-2.1.8-sources.jar	Updated	org.jfrog.test:multi1:2.1.8	
multi1-2.1.8.jar	multi1-2.1.8.jar	Updated	org.jfrog.test:multi1:2.1.8	
multi1-2.1.8.pom	multi1-2.1.8.pom	Updated	org.jfrog.test:multi1:2.1.8	http://10.100.1.110:8081/artifacto...
multi3-2.1.8.pom	multi3-2.1.8.pom	Updated	org.jfrog.test:multi3:2.1.8	http://10.100.1.110:8081/artifacto...

< page 1 of 1 >

Release History

The Release History tab displays a list of the selected build's release landmarks.

All Builds > maven-example > 60

Build Browser

Build #60

General Build Info | Published Modules | Environment | Issues | Licenses | Governance | **Release History** >>

Repository:	ext-release-local
Comment:	Staging version 2.1.8
CI User:	eli
Artifactory User:	jenkins

Build Info JSON

Generic BuildInfo View

This tab displays the raw BuildInfo JSON representation of the build information in Artifactory. This data can be accessed via the REST API or used for debugging and is also available in the Artifactory OSS version.

All Builds > maven-example > 60

Build Browser

Build #60

General Build Info | Published Modules | Environment | Issues | Licenses | Governance | **Build Info JSON** >>

Build Info JSON

```

1 {
2   "properties" : {
3     "java.vendor" : "Apple Inc.",
4     "buildInfo.env.USER" : "eli",
5     "sun.java.launcher" : "SUN_STANDARD",
6     "buildInfo.env.BUILD_TAG" : "jenkins-maven-example-60",
7     "sun.management.compiler" : "HotSpot 64-Bit Tiered Compilers",
8     "os.name" : "Mac OS X",
9     "buildInfo.env.PATH" :
10    "/usr/share/maven/bin:/System/Library/Frameworks/JavaVM.Framework/Versions/1.6.0/Home/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin:/usr/X11/bin",
11    "buildInfo.env.LOGNAME" : "eli",
12    "sun.boot.class.path" :
13    "/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Classes/jsfd.jar:/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Classes/classes.jar:/System/Library/Frameworks/JavaVM.Framework/Frameworks/JavaRuntimeSupport.framework/Resources/Java/JavaRuntimeSupport.jar:/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Classes/ui.jar:/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Classes/laf.jar:/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Classes/sunrsasign.jar:/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Classes/jsse.jar:/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Classes/jce.jar:/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Classes/charsets.jar",
14    "java.vm.specification.vendor" : "Sun Microsystems Inc.",
15    "java.runtime.version" : "1.6.0_33-b03-424-11M3720",
16    "user.name" : "eli",
17    "buildInfo.env.buildInfoConfig.propertiesFile" : "/usr/local/Cellar/tomcat/7.0.29/libexec/temp/buildInfo3361433366778964639.properties",
18    "guice.disable.misplaced.annotation.check" : "true",
19    "awt.nativeDoubleBuffering" : "true",
20    "m3plugin.lib" : "/Users/eli/.jenkins/plugins/artifactory-2.1.3-SNAPSHOT/WEB-INF/lib",
21    "buildInfo.env.BUILD_URI" : "http://repo-demo:8080/jenkins/job/maven-example/60/"
22  }
23 }

```

Exporting and Manipulating Build Items

You can view a build in the repository browser and perform actions on it as a whole with all its artifacts and dependencies. For example, you could promote it to another repository, copy it, or export it to a disk.

The screenshot shows the 'Artifact Repository Browser' interface. On the left, a tree view shows the artifact's location: `ext-release-local/org/jfrog/test/multi1/maven-metadata.xml`. The main panel displays the 'maven-metadata.xml' artifact details under the 'General' tab. The 'Info' section includes:

- Name: maven-metadata.xml
- Repository Path: ext-release-local:org/jfrog/test/multi1/maven-metadata.xml
- Created: 30-04-15 11:30:58 UTC
- Deployed by: admin
- Size: 383 bytes
- Last Modified: 08-07-15 09:38:43 UTC
- Module ID: N/A
- Licenses: Not Found (Add | Scan)
- Downloaded: 1
- Last Downloaded: 22-06-15 11:39:54 UTC
- Last Downloaded by: admin

The 'Dependency Declaration' section shows the artifact is used by Maven, with the following XML snippet:

```

1 <dependency>
2   <groupId>org.jfrog</groupId>
3   <artifactId>test</artifactId>
4   <version>multi1</version>
5   <type>ta.xml</type>
6 </dependency>

```

Repository View of Builds

When viewing an artifact within the **Tree Browser**, you can see all of the builds with which that artifact is associated, whether directly or as a dependency in the **Builds** tab

Moreover, if you try to remove the artifact you will receive a warning that the build will no longer be reproducible.

The screenshot shows the 'Artifact Repository Browser' interface with the 'Builds' tab selected for the artifact `multi1-2.18-20150625.110536-1.pom`. The 'Produced By' section shows a table of builds that produced this artifact:

Build Name	Build Number	Module ID	Started at	CI Server
mavenProject-NOTfreest...	10	org.jfrog.test:multi1:2.18-...	1435230341754	http://10.0.0.134:8080/jo...
mavenProject-NOTfreest...	14	org.jfrog.test:multi1:2.18-...	1435230542843	http://10.0.0.134:8080/jo...
mavenProject-NOTfreest...	11	org.jfrog.test:multi1:2.18-...	1435230392479	http://10.0.0.134:8080/jo...
mavenProject-NOTfreest...	13	org.jfrog.test:multi1:2.18-...	1435230493924	http://10.0.0.134:8080/jo...
mavenProject-NOTfreest...	12	org.jfrog.test:multi1:2.18-...	1435230443839	http://10.0.0.134:8080/jo...

The association of an artifact with a build is retained even if you move or copy it within Artifactory, because the association linked to the artifact's checksum which remains constant, regardless of its location.

Behind the Scenes

Behind the scenes, the Artifactory plug-in for your CI server performs two major tasks:

1. It resolves all dependencies from a resolution repository in Artifactory.
2. It deploys all the artifacts to Artifactory as an atomic operation at the end of the build, guaranteeing a more coherent deployment when building multi-module projects (Maven and Ivy deploy each module at the end of its build cycle. If one of the modules fails, this can result in partial deployments).
3. It sends a `BuildInfo` data object to Artifactory via the REST API at the end of deployment. This is a structured JSON object containing all the data about the build environment, artifacts and dependencies, in a standard and open format.

You can find the latest `BuildInfo` Java-binding artifacts [here](#) and the source [here](#).

Release Management

Artifactory supports release management through its plugins for [Jenkins](#), [TeamCity](#) and [Bamboo](#).

When you run your builds using [Maven](#) or [Gradle](#) with jobs that use [Subversion](#), [Git](#) or [Perforce](#) as your version control system, you can manually stage a release build allowing you to:

- Change values for the release and next development version
- Choose a target staging repository to which to deploy the release
- Create a VCS tag for the release

Staged release builds can later be **promoted** or **rolled-back**, changing their release status in Artifactory, with the option to move the build artifacts to a different target repository.

Inside Artifactory, the history of all build status change activities (staged, promoted, rolled-back, etc.) is recorded and displayed for full traceability.

To learn more about release management specific to your CI server, please refer to:

★ [Release Management in the Jenkins Documentation](#)

★ [TeamCity Artifactory Plugin - Release Management](#)

★ [Bamboo Artifactory Plug-in - Release Management](#)

Jenkins Artifactory Plug-in

Overview

Artifactory provides tight integration with Jenkins a plugin which you need to install using Jenkins Plugin Manager. For more information, please refer to the [Jenkins documentation](#).

The plug-in provides:

- Easy setup to resolve dependencies and deploy build artifacts through Artifactory.
- Capture exhaustive build information such as artifacts deployed, dependencies resolved, system and environment information and more to enable fully traceable builds.
- Enhanced deployment that transfers additional important build information to Artifactory.
- UI integration providing links from a Jenkins build directly to Artifactory.
- Release management with staging and promotion.
- Extensive APIs for Pipeline jobs.

Before you begin

Please refer to the general information about [Artifactory's Build Integration](#) before using the

Jenkins Artifactory Plugin.

Source Code Available!

The Jenkins Artifactory Plugin is an [open source project on GitHub](#) which you can freely browse and fork.

Supported Build Technologies

The Jenkins Artifactory Plugin currently supports [Maven 3](#), [Maven2](#), [Gradle](#), and [Ivy/Ant](#), as well as [generic \(free style\)](#) builds that use other build tools.

Page Contents

- [Overview](#)
 - [Supported Build Technologies](#)
- [Installing and Configuring the Plug-in](#)
- [Navigating Between Jenkins and Artifactory](#)
- [Watch the Screencast](#)

Read more

- [Working With Pipeline Jobs in Jenkins](#)

Installing and Configuring the Plug-in

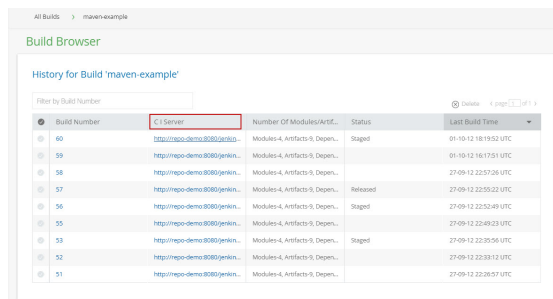
For information about installing and configuring the plug-in, please refer to the instructions at the [Jenkins Plug-in Center](#).

Navigating Between Jenkins and Artifactory

Each build viewed in Artifactory corresponds to its Jenkins job counterpart, and contains a build history of all runs corresponding to the build history in Jenkins.

You can [drill down and view details](#) about each item in the build history, or view the build in the CI server by selecting the corresponding link in the **CI Server** column.

You can also navigate back to Artifactory from Jenkins, as shown below:



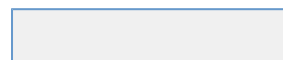
The screenshot shows the Jenkins Build Browser interface for a job named 'maven-example'. It displays a table of build history with columns for Build Number, CI Server, Number of Modules/Artifactory, Status, and Last Build Time. The 'CI Server' column contains links to the corresponding build in Jenkins. The 'Status' column shows various states like 'Staged' and 'Released'. The 'Last Build Time' column shows the timestamp of the build in UTC.

Build Number	CI Server	Number of Modules/Artifactory	Status	Last Build Time
60	http://maven-example@000jenkins...	Modules-4, Artifactory-9, Depen...	Staged	01-10-12 18:19:52 UTC
59	http://maven-example@000jenkins...	Modules-4, Artifactory-9, Depen...	Staged	01-10-12 16:17:51 UTC
58	http://maven-example@000jenkins...	Modules-4, Artifactory-9, Depen...	Released	27-09-12 22:57:26 UTC
57	http://maven-example@000jenkins...	Modules-4, Artifactory-9, Depen...	Released	27-09-12 22:55:22 UTC
56	http://maven-example@000jenkins...	Modules-4, Artifactory-9, Depen...	Staged	27-09-12 22:52:49 UTC
55	http://maven-example@000jenkins...	Modules-4, Artifactory-9, Depen...	Staged	27-09-12 22:49:23 UTC
53	http://maven-example@000jenkins...	Modules-4, Artifactory-9, Depen...	Staged	27-09-12 22:39:56 UTC
52	http://maven-example@000jenkins...	Modules-4, Artifactory-9, Depen...	Staged	27-09-12 22:38:12 UTC
51	http://maven-example@000jenkins...	Modules-4, Artifactory-9, Depen...	Staged	27-09-12 22:28:57 UTC

Watch the Screencast

Working With Pipeline Jobs in Jenkins

Introduction



The Pipeline Jenkins Plugin simplifies building a continuous delivery pipeline with Jenkins by creating a script that defines the steps of your build. For those not familiar with Jenkins Pipeline, please refer to the [Pipeline Tutorial](#) or the [Getting Started With Pipeline](#) documentation.

The [Jenkins Artifactory Plugin](#) has been extended to support Artifactory operations as part of the Pipeline script DSL. You have the added option of downloading dependencies, uploading artifacts, and publishing build-info to Artifactory from a Pipeline script.

Using the Artifactory DSL

Creating an Artifactory Server Instance

To upload or download files to and from your Artifactory server, you need to create an Artifactory server instance in your Pipeline script.

If your Artifactory server is already defined in Jenkins, you only need its server ID which can be obtained under **Manage | Configure System**.

Then, to create your Artifactory server instance, add the following line to your script:

```
def server = Artifactory.server 'my-server-id'
```

If your Artifactory is not defined in Jenkins you can still create it as follows:

```
def server = Artifactory.newServer url:
'artifactory-url', username: 'username',
password: 'password'
```

You can also use Jenkins Credential ID instead of username and password:

```
def server = Artifactory.newServer url:
'artifactory-url', credentialsId:
'ccrreeddeennttiaall'
```

You can modify the server object using the following methods:

```
server.bypassProxy = true
// If you're using username and password:
server.username = 'new-user-name'
server.password = 'new-password'
// If you're using Credentials ID:
server.credentialsId = 'ccrreeddeennttiaall'
```

Use variables

We recommend using variables rather than plain text to specify the Artifactory server details.

Page contents

- Introduction
- Using the Artifactory DSL
 - Creating an Artifactory Server Instance

- Uploading and Downloading Files
 - Publishing Build-Info to Artifactory
 - Promoting Builds in Artifactory
 - Allowing Interactive Promotion for Published Builds
 - Maven Builds with Artifactory
 - Gradle Builds with Artifactory
 - Maven Release Management with Artifactory
 - Conan Builds with Artifactory
 - Docker Builds with Artifactory
 - Scanning Builds with JFrog Xray
 - Distributing Build Artifacts
- File Spec Schema
 - Examples

Uploading and Downloading Files

To upload or download files you first need to create a spec which is a JSON file that specifies which files should be uploaded or downloaded and the target path.

For example:

```
def downloadSpec = """{
  "files": [
    {
      "pattern": "bazinga-repo/*.zip",
      "target": "bazinga/"
    }
  ]
}"""
```

The above spec specifies that all ZIP files in the *bazinga-repo* Artifactory repository should be downloaded into the *bazinga* directory on your Jenkins agent file system.

"files" is an array

Since the "files" element is an array, you can specify several patterns and corresponding targets in a single download spec.

To download the files, add the following line to your script:

```
server.download(downloadSpec)
```

Uploading files is very similar. The following example uploads all ZIP files that include *froggy* in their names into the *froggy-files* folder in the *bazinga-repo* Artifactory repository.

```
def uploadSpec = """{
  "files": [
    {
      "pattern": "bazinga/*froggy*.zip",
      "target": "bazinga-repo/froggy-files/"
    }
  ]
}"""
server.upload(uploadSpec)
```

You can read about using File Specs for downloading and uploading files [here](#).

Publishing Build-Info to Artifactory

Both the download and upload methods return a build-info object which can be published to Artifactory as shown in the following examples:

```
def buildInfo1 = server.download downloadSpec
def buildInfo2 = server.upload uploadSpec
buildInfo1.append buildInfo2
server.publishBuildInfo buildInfo1
```

```
def buildInfo = Artifactory.newBuildInfo()
server.download spec: downloadSpec, buildInfo: buildInfo
server.upload spec: uploadSpec, buildInfo: buildInfo
server.publishBuildInfo buildInfo
```

Modifying the Default Build Name and Build Number

You can modify the default build name and build number set by Jenkins. Here's how you do it:

```
def buildInfo = Artifactory.newBuildInfo()
buildInfo.name = 'super-frog'
buildInfo.number = 'v1.2.3'
server.publishBuildInfo buildInfo
```

If you're setting the build name or number as shown above, it is important to do so before you're using this buildInfo instance for uploading files.

Here's the reason for this: The server.upload method also tags the uploaded files with the build name and build number (using the [build.name](#) and build.number properties). Setting a new build name or number on the buildInfo instance will not update the properties attached to the files.

Capturing Environment Variables

To set the Build-Info object to automatically capture environment variables while downloading and uploading files, add the following to your script:

```
def buildInfo = Artifactory.newBuildInfo()
buildInfo.env.capture = true
```

By default, environment variables named "password", "secret", or "key" are excluded and will not be published to Artifactory.

You can add more include/exclude patterns as follows:

```
def buildInfo = Artifactory.newBuildInfo()
buildInfo.env.filter.addInclude( "*"a*" )
buildInfo.env.filter.addExclude( "DONT_COLLECT*" )
```

Here's how you reset to the include/exclude patterns default values:

```
buildInfo.env.filter.reset()
```

You can also completely clear the include/exclude patterns:

```
buildInfo.env.filter.clear()
```

To collect environment variables at any point in the script, use:

```
buildInfo.env.collect()
```

You can get the value of an environment variable collected as follows:

```
value = buildInfo.env.vars['env-var-name']
```

Triggering Build Retention

To trigger build retention when publishing build-info to Artifactory, use the following method:

```
buildInfo.retention maxBuilds: 10
```

```
buildInfo.retention maxDays: 7
```

To have the build retention also delete the build artifacts, add the **deleteBuildArtifacts** with **true** value as shown below:

```
buildInfo.retention maxBuilds: 10, maxDays: 7, doNotDiscardBuilds: ["3",  
"4"], deleteBuildArtifacts: true
```

It is possible to trigger an asynchronous build retention. To do this, add the **async** argument with **true** as shown below:

```
buildInfo.retention maxBuilds: 10, deleteBuildArtifacts: true, async:  
true
```

Promoting Builds in Artifactory

To promote a build between repositories in Artifactory, define the promotion parameters in a promotionConfig object and promote that. For example:

```

def promotionConfig = [
    // Mandatory parameters
    'buildName'      : buildInfo.name,
    'buildNumber'   : buildInfo.number,
    'targetRepo'    : 'libs-release-local',

    // Optional parameters
    'comment'       : 'this is the promotion comment',
    'sourceRepo'    : 'libs-snapshot-local',
    'status'        : 'Released',
    'includeDependencies': true,
    'copy'          : true,
    // 'failFast' is true by default.
    // Set it to false, if you don't want the promotion to abort
    upon receiving the first error.
    'failFast'      : true
]

// Promote build
server.promote promotionConfig

```

Allowing Interactive Promotion for Published Builds

The 'Promoting Builds in Artifactory' section in this article describes how your Pipeline script can promote builds in Artifactory. In some cases however, you'd like the build promotion to be performed after the build finished. You can configure your Pipeline job to expose some or all the builds it publishes to Artifactory, so that they can be later promoted interactively using a GUI. Here's how the Interactive Promotions looks like:

When the build finishes, the promotion window will be accessible by clicking on the promotion icon, next to the build run.

Here's how you do this.

First you need to create a 'promotionConfig' instance, the same way it is shown in the 'Promoting Builds in Artifactory' section.

Next, you can use it, to expose a build for interactive promotion as follows:

```

Artifactory.addInteractivePromotion server: server, promotionConfig:
promotionConfig, displayName: "Promote me please"

```

You can add as many builds as you like, by using the method multiple times. All the builds added will be displayed in the promotion window.

The 'addInteractivePromotion' methods expects the following arguments:

1. "server" is the Artifactory on which the build promotions is done. You can create the server instance as described in the beginning of this article.
2. "promotionConfig" includes the promotion details. The "Promoting Builds in Artifactory" section describes how to create a promotionConfig instance.
3. "displayName" is an optional argument. If you add it, the promotion window will display it instead of the build name and number.

Maven Builds with Artifactory

Maven builds can resolve dependencies, deploy artifacts and publish build-info to Artifactory. To run Maven builds with Artifactory from your Pipeline script, you first need to create an Artifactory server instance, as described at the beginning of this article. Here's an example:

```
def server = Artifactory.server('my-server-id')
```

The next step is to create an Artifactory Maven Build instance:

```
def rtMaven = Artifactory.newMavenBuild()
```

Now let's define where the Maven build should download its dependencies from. Let's say you want the release dependencies to be resolved from the 'libs-release' repository and the snapshot dependencies from the 'libs-snapshot' repository. Both repositories are located on the Artifactory server instance you defined above. Here's how you define this, using the Artifactory Maven Build instance we created:

```
rtMaven.resolver server: server, releaseRepo: 'libs-release',  
snapshotRepo: 'libs-snapshot'
```

Now let's define where our build artifacts should be deployed to. Once again, we define the Artifactory server and repositories on the 'rtMaven' instance:

```
rtMaven.deployer server: server, releaseRepo: 'libs-release-local',  
snapshotRepo: 'libs-snapshot-local'
```

By default, all the build artifacts are deployed to Artifactory. In case you want to deploy only some artifacts, you can filter them based on their names, using the 'addInclude' method. In the following example, we are deploying only artifacts with names that start with 'frog'

```
rtMaven.deployer.artifactDeploymentPatterns.addInclude("frog*")
```

You can also exclude artifacts from being deployed. In the following example, we are deploying all artifacts, except for those that are zip files:

```
rtMaven.deployer.artifactDeploymentPatterns.addExclude("*.zip")
```

And to make things more interesting, you can combine both methods. For example, to deploy all artifacts with names that start with 'frog', but are not zip files, do the following:

```
rtMaven.deployer.artifactDeploymentPatterns.addInclude("frog*").addExclude("*.zip")
```

If you'd like to add custom properties to the deployed artifacts, you can do that as follows:

```
rtMaven.deployer.addProperty("status",  
"in-qa").addProperty("compatibility", "1", "2", "3")
```

In some cases, you want to disable artifacts deployment to Artifactory or make the deployment conditional. Here's how you do it:

```
rtMaven.deployer.deployArtifacts = false
```

To select a Maven installation for our build, we should define a Maven Tool through Jenkins Manage, and then, set the tool name as follows:

```
rtMaven.tool = 'maven tool name'
```

Here's how you define Maven options for your build:

```
rtMaven.opts = '-Xms1024m -Xmx4096m'
```

In case you'd like Maven to use a different JDK than your build agent's default, no problem.

Simply set the `JAVA_HOME` environment variable to the desired JDK path (the path to the directory above the bin directory, which includes the java executable).

Here's you do it:

```
env.JAVA_HOME = 'path to JDK'
```

OK, we're ready to run our build. Here's how we define the pom file path (relative to the workspace) and the Maven goals. The deployment to Artifactory is performed during the 'install' phase:

```
def buildInfo = rtMaven.run pom: 'maven-example/pom.xml', goals: 'clean install'
```

The above method runs the Maven build.

By default, the build artifacts will be deployed to Artifactory, unless `rtMaven.deployer.deployArtifacts` property was set to false.

In this case, artifacts can be deployed using the following step:

```
rtMaven.deployer.deployArtifacts buildInfo
```

Make sure to use the same `buildInfo` instance you received from the `rtMaven.run` method. Also make sure to run the above method on the same agent that ran the `rtMaven.run` method, because the artifacts were built and stored on the file-system of this agent.

By default, Maven uses the local Maven repository inside the `.m2` directory under the user home. In case you'd like Maven to create the local repository in your job's workspace, add the `-Dmaven.repo.local=.m2` system property to the goals value as shown here:

```
def buildInfo = rtMaven.run pom: 'maven-example/pom.xml', goals: 'clean install -Dmaven.repo.local=.m2'
```

What about the build information?

The build information has not yet been published to Artifactory, but it is stored locally in the 'buildInfo' instance returned by the 'run' method.

You can now publish it to Artifactory as follows:

```
server.publishBuildInfo buildInfo
```

You can also merge multiple `buildInfo` instances into one `buildInfo` instance and publish it to Artifactory as one build, as described in the 'Publishing Build-Info to Artifactory' section in this article.

Gradle Builds with Artifactory

Gradle builds can resolve dependencies, deploy artifacts and publish build-info to Artifactory. To run Gradle builds with Artifactory from your Pipeline script, you first need to create an Artifactory server instance, as described at the beginning of this article.

Here's an example:

```
def server = Artifactory.server 'my-server-id'
```

The next step is to create an Artifactory Gradle Build instance:

```
def rtGradle = Artifactory.newGradleBuild()
```

Now let's define where the Gradle build should download its dependencies from. Let's say you want the dependencies to be resolved from the 'libs-release' repository, located on the Artifactory server instance you defined above. Here's how you define this, using the Artifactory Gradle Build instance we created:

```
rtGradle.resolver server: server, repo: 'libs-release'
```

Now let's define where our build artifacts should be deployed to. Once again, we define the Artifactory server and repositories on the 'rtGradle' instance:

```
rtGradle.deployer server: server, repo: 'libs-release-local'
```

By default, all the build artifacts are deployed to Artifactory. In case you want to deploy only some artifacts, you can filter them based on their names, using the 'addInclude' method. In the following example, we are deploying only artifacts with names that start with 'frog'

```
rtGradle.deployer.artifactDeploymentPatterns.addInclude("frog*")
```

You can also exclude artifacts from being deployed. In the following example, we are deploying all artifacts, except for those that are zip files:

```
rtGradle.deployer.artifactDeploymentPatterns.addExclude("*.zip")
```

And to make things more interesting, you can combine both methods. For example, to deploy all artifacts with names that start with 'frog', but are not zip files, do the following:

```
rtGradle.deployer.artifactDeploymentPatterns.addInclude("frog*").addExclude("*.zip")
```

If you'd like to add custom properties to the deployed artifacts, you can do that as follows:

```
rtGradle.deployer.addProperty("status",  
"in-qa").addProperty("compatibility", "1", "2", "3")
```

In some cases, you want to disable artifacts deployment to Artifactory or make the deployment conditional. Here's how you do it:

```
rtGradle.deployer.deployArtifacts = false
```

In case the "com.jfrog.artifactory" Gradle Plugin is already applied in your Gradle script, we need to let Jenkins know it shouldn't apply it. Here's how we do it:


```
rtGradle.usesPlugin = true
```

In case you'd like to use the Gradle Wrapper for this build, add this:

```
rtGradle.useWrapper = true
```

If you don't want to use the Gradle Wrapper, and set a Gradle installation instead, you should define a Gradle Tool through Jenkins Manage, and then, set the tool name as follows:

```
rtGradle.tool = 'gradle tool name'
```

In case you'd like Maven to use a different JDK than your build agent's default, no problem.

Simply set the JAVA_HOME environment variable to the desired JDK path (the path to the directory above the bin directory, which includes the java executable).

Here's you do it:

```
env.JAVA_HOME = 'path to JDK'
```

OK, looks like we're ready to run our Gradle build. Here's how we define the build.gradle file path (relative to the workspace) and the Gradle tasks. The deployment to Artifactory is performed as part of the 'artifactoryPublish' task:

```
def buildInfo = rtGradle.run rootDir: "projectDir/", buildFile:  
'build.gradle', tasks: 'clean artifactoryPublish'
```

The above method runs the Gradle build.

By default, the build artifacts will be deployed to Artifactory, unless `rtGradle.deployer.deployArtifacts` property was set to false.

In this case, artifacts can be deployed using the following step:

```
rtGradle.deployer.deployArtifacts buildInfo
```

Make sure to use the same buildInfo instance you received from the `rtMaven.run` method. Also make sure to run the above method on the same agent that ran the `rtMaven.run` method, because the artifacts were built and stored on the file-system of this agent.

What about the build information?

The build information has not yet been published to Artifactory, but it is stored locally in the 'buildInfo' instance returned by the 'run' method. You can now publish it to Artifactory as follows:

```
server.publishBuildInfo buildInfo
```

You can also merge multiple buildInfo instances into one buildInfo instance and publish it to Artifactory as one build, as described in the 'Publishing Build-Info to Artifactory' section in this article.

That's it! We're all set.

The `rtGradle` instance supports additional configuration APIs. You can use these APIs as follows:

```

def rtGradle = Artifactory.newGradleBuild()
// Deploy Maven descriptors to Artifactory:
rtGradle.deployer.deployMavenDescriptors = true
// Deploy Ivy descriptors (pom.xml files) to Artifactory:
rtGradle.deployer.deployIvyDescriptors = true

// The following properties are used for Ivy publication
configuration.
// The values below are the defaults.

// Set the deployed Ivy descriptor pattern:
rtGradle.deployer.ivyPattern =
'[organisation]/[module]/ivy-[revision].xml'
// Set the deployed Ivy artifacts pattern:
rtGradle.deployer.artifactPattern =
'[organisation]/[module]/[revision]/[artifact]-[revision](-[classifier])
.[ext]'
// Set mavenCompatible to true, if you wish to replace dots with
slashes in the Ivy layout path, to match the Maven layout:
rtGradle.deployer.mavenCompatible = true

```

Maven Release Management with Artifactory

With the Artifactory Pipeline DSL you can easily manage and run a release build for your Maven project by following the instructions below:

First, clone the code from your source control:

```
git url: 'https://github.com/eyalbe4/project-examples.git'
```

If the pom file has a snapshot version, Maven will create snapshot artifacts, because the pom files include a snapshot version (for example, 1.0.0-SNAPSHOT).

Since you want your build to create release artifacts, you need to change the version in the pom file to 1.0.0.

To do that, create a mavenDescriptor instance, and set the version to 1.0.0:

```
def descriptor = Artifactory.mavenDescriptor()
descriptor.version = '1.0.0'
```

If the project's pom file is not located at the root of the cloned project, but inside a sub-directory, add it to the mavenDescriptor instance:

```
descriptor.pomFile = 'maven-example/pom.xml'
```

In most cases, you want to verify that your release build does not include snapshot dependencies. There are two ways to do that.

The first way, is to configure the descriptor to fail the build if snapshot dependencies are found in the pom files. In this case, the job will fail before the new version is set to the pom files.

Here's how you configure this:

```
descriptor.failOnSnapshot = true
```

The second way to verify this is by using the hasSnapshots method, which returns a boolean true value if snapshot dependencies are found:

```
def snapshots = descriptor.hasSnapshots()
  if (snapshots) {
    ....
  }
```

That's it. Using the `mavenDescriptor` as it is now will change the version inside the root pom file. In addition, if the project includes sub-modules with pom files, which include a version, it will change them as well. Sometimes however, some sub-modules should use different release versions. For example, suppose there's one module whose version should change to 1.0.1, instead of 1.0.0. The other modules should still have their versions changed to 1.0.0. Here's how to do that:

```
descriptor.setVersion "the.group.id:the.artifact.id", "1.0.1"
```

The above `setVersion` method receives two arguments: the module name and its new release version. The module name is composed of the group ID and the artifact ID with a colon between them. Now you can transform the pom files to include the new versions:

```
descriptor.transform()
```

The `transform` method changed the versions on the local pom files. You can now build the code and deploy the release Maven artifacts to Artifactory as described in the "[Maven Builds with Artifactory](#)" section in this article.

The next step is to commit the changes made to the pom files to the source control, and also tag the new release version in the source control repository. If you're using git, you can use the git client installed on your build agent and run a few shell commands from inside the Pipeline script to do that.

The last thing you'll probably want to do is to change the pom files version to the next development version and commit the changes. You can do that again by using a `mavenDescriptor` instance.

Conan Builds with Artifactory

[Conan](#) is a C/C++ Package Manager. The Artifactory Pipeline DSL includes APIs that make it easy for you to run Conan builds, using the Conan Client installed on your build agents. Here's what you need to do before you create your first Conan build job with Jenkins:

1. Install the latest Conan Client on your Jenkins build agent. Please refer to the [Conan documentation](#) for installation instructions.
2. Add the Conan Client executable to the PATH environment variable on your build agent, to make sure Jenkins is able to use the client.
3. Create a Conan repository in Artifactory as described in the [Conan Repositories Artifactory documentation](#).

OK. Let's start coding your first Conan Pipeline script.

We'll start by creating an Artifactory server instance, as described at the beginning of this article. Here's an example:

```
def server = Artifactory.server 'my-server-id'
```

Now let's create a Conan Client instance

```
def conanClient = Artifactory.newConanClient()
```

When creating the Conan client, you can also specify the Conan user home directory as shown below:

```
def conanClient = Artifactory.newConanClient userHome:
  "conan/my-conan-user-home"
```

We can now configure our new `conanClient` instance by adding an Artifactory repository to it. In our example, we're adding the 'conan-local' repository, located in the Artifactory server, referenced by the server instance we obtained:

```
String remoteName = conanClient.remote.add server: server, repo:
"conan-local"
```

As you can see in the above example, the `'conanClient.remote.add'` method returns a string variable - `'remoteName'`.

What is this `'remoteName'` variable? What is it for?

Well, a 'Conan remote' is a repository, which can be used to download dependencies from and upload artifacts to. When we added the 'conan-local' Artifactory repository to our Conan Client, we actually added a Conan remote. The `'remoteName'` variable contains the name of the new Conan remote we added.

OK. We're ready to start running Conan commands. You'll need to be familiar with the Conan commands syntax, exposed by the Conan Client to run the commands. You can read about the commands syntax in the [Conan documentation](#).

Let's run the first command:

```
def buildInfo1 = conanClient.run command: "install --build
missing"
```

The `'conanClient.run'` method returns a `buildInfo` instance, that we can later publish to Artifactory. If you already have a `buildInfo` instance, and you'd like the `'conanClient.run'` method to aggregate the build information to it, you can also send the `buildInfo` instance to the run command as an argument as shown below:

```
conanClient.run command: "install --build missing", buildInfo:
buildInfo
```

The next thing we want to do is to use the Conan remote we created. For example, let's upload our artifacts to the Conan remote. Notice how we use the `'remoteName'` variable we got earlier, when building the Conan command:

```
String command = "upload * --all -r ${remoteName}
--confirm"
conanClient.run command: command, buildInfo: buildInfo
```

We can now publish the `buildInfo` to Artifactory, as described in the 'Publishing Build-Info to Artifactory' section in this article. For example:

```
server.publishBuildInfo buildInfo
```

Docker Builds with Artifactory

The Jenkins Artifactory Plugin supports a Pipeline DSL that enables you to collect and publish build-info to Artifactory for your Docker builds. To collect the build-info, the plugin uses an internal HTTP proxy server, which captures the traffic between the Docker Daemon and your Artifactory reverse proxy. To setup your Jenkins build agents to collect build-info for your Docker builds, please refer to the [setup instructions](#)

```

// Create an Artifactory server instance, as described above in this
article:
    def server = Artifactory.server 'my-server-id'

    // Create an Artifactory Docker instance. The instance stores the
    Artifactory credentials and the Docker daemon host address:
    def rtDocker = Artifactory.docker username: 'artifactory-username',
password: 'artifactory-password', host: "tcp://<daemon IP>:<daemon
port>"

    // If the docker daemon host is not specified,
"/var/run/dokcer.sock" is used as a default value:
    def rtDocker = Artifactory.docker username: 'artifactory-username',
password: 'artifactory-password'

    // You can also use the Jenkins credentials plugin instead of
username and password:
    def rtDocker = Artifactory.docker credentialsId:
'ccrreeddeennttiiaall'

    // Attach custom properties to the published artifacts:
    rtDocker.addProperty("project-name",
"docker1").addProperty("status", "stable")

    // Push a docker image to Artifactory (here we're pushing
hello-world:latest). The push method also expects
    // Artifactory repository name:
    def buildInfo =
rtDocker.push('<artifactory-docker-registry-url>/hello-world:latest',
'<target-artifactory-repository>')

    // Publish the build-info to Artifactory:
    server.publishBuildInfo buildInfo

```

Scanning Builds with JFrog Xray

From version 2.9.0, Jenkins Artifactory Plugin is integrated with JFrog Xray through JFrog Artifactory allowing you to have build artifacts and dependencies scanned for vulnerabilities and other issues. If issues or vulnerabilities are found, you may choose to fail a build job or perform other actions according to the Pipeline script you write. This integration requires **JFrog Artifactory v4.16** and above and **JFrog Xray v1.6** and above.

You may scan any build that has been published to Artifactory. It does not matter when the build was published, as long as it was published before triggering the scan by JFrog Xray.

The following instructions show you how to configure your Pipeline script to have a build scanned.

First, create a scanConfig instance with the build name and build number you wish to scan:

```

def scanConfig = [
    'buildName'      : 'my-build-name',
    'buildNumber'   : '17'
]

```

If you're scanning a build which has already been published to Artifactory in the same job, you can use the build name and build number stored on the `buildInfo` instance you used to publish the build. For example:

```
server.publishBuildInfo buildInfo
  def scanConfig = [
    'buildName'      : buildInfo.name,
    'buildNumber'    : buildInfo.number
  ]
```

Before you trigger the scan, there's one more thing you need to be aware of. By default, if the Xray scan finds vulnerabilities or issues in the build that trigger an alert, the build job will fail. If you don't want the build job to fail, you can add the `'failBuild'` property to the `scanConfig` instance and set it to `'false'` as shown here:

```
def scanConfig = [
  'buildName'      : buildInfo.name,
  'buildNumber'    : buildInfo.number,
  'failBuild'      : false
]
```

OK, we're ready to initiate the scan. The scan should be initiated on the same Artifactory server instance, to which the build was published:

```
def scanResult = server.xrayScan scanConfig
```

That's it. The build will now be scanned. If the scan is not configured to fail the build job, you can use the `scanResult` instance returned from the `xrayScan` method to see some details about the scan.

For example, to print the result to the log, you could use the following code snippet:

```
echo scanResult as String
```

For more details on the integration with JFrog Xray and JFrog Artifactory to scan builds for issues and vulnerabilities, please refer to [CI/CD Integration](#) in the JFrog Xray documentation.

Distributing Build Artifacts

From version 2.11.0, you can easily distribute your build artifacts to the world or to your community using Pipeline. Your artifacts are distributed to JFrog Bintray, using a Distribution Repository in Artifactory. You can read more about Distribution Repositories and the steps for setting them up [here](#).

In order to distribute a build, it should be first published to Artifactory, as described in the "Publishing Build-Info to Artifactory" section in this article.

Once you have your Distribution Repository set up and your build is published to Artifactory, define a `distributionConfig` instance in your Pipeline script as shown here:

```
def distributionConfig = [  
  // Mandatory parameters  
  'buildName'           : buildInfo.name,  
  'buildNumber'        : buildInfo.number,  
  'targetRepo'         : 'dist-repo',  
  
  // Optional parameters  
  'publish'            : true, // Default: true. If true, artifacts  
                           are published when deployed to Bintray.  
  'overrideExistingFiles' : false, // Default: false. If true,  
                           Artifactory overwrites builds already existing in the target path in  
                           Bintray.  
  'gpgPassphrase'      : 'passphrase', // If specified, Artifactory  
                           will GPG sign the build deployed to Bintray and apply the specified  
                           passphrase.  
  'async'              : false, // Default: false. If true, the  
                           build will be distributed asynchronously. Errors and warnings may be  
                           viewed in the Artifactory log.  
  'sourceRepos'        : ["yum-local"], // An array of local  
                           repositories from which build artifacts should be collected.  
  'dryRun'             : false, // Default: false. If true,  
                           distribution is only simulated. No files are actually moved.  
]
```

OK, we're ready to distribute the build. The distribution should be done on the same Artifactory server instance, to which the build was published:

```
server.distribute distributionConfig
```

File Spec Schema

You can read the File Spec schema [here](#).

Examples

The [Jenkins Pipeline Examples](#) can help get you started using the Artifactory DSL in your Pipeline scripts.

TeamCity Artifactory Plug-in

Overview

Artifactory provides tight integration with TeamCity CI Server through the TeamCity Artifactory Plug-in. Beyond managing efficient deployment of your artifacts to Artifactory, the plug-in lets you capture information about artifacts deployed, dependencies resolved, environment data associated with the TeamCity build runs and more, that effectively provides full traceability for your builds.

From version 2.1.0 the TeamCity Artifactory Plug-in provides powerful features for release management and promotion. For details please refer to [TeamCity Artifactory Plug-in - Release Management](#).

Before you begin

Please refer to the general information about [Artifactory's Build Integration](#) before using the TeamCity Artifactory Plugin.

Source Code Available!

The TeamCity Artifactory Plugin is an open source project on GitHub which you can freely browse and fork.

Build Runner Support

The TeamCity Artifactory plugin supports most build runner types, including: **Maven2**, **Maven 3**, **Ivy/Ant** (with Ivy modules support), **Gradle**, **NAnt**, **MSBuild**, **FxCop** and **Ipr**.

Page Contents

- Overview
 - Build Runner Support
- Installing the Plugin
- Configuration
 - Configuring System-wide Artifactory Servers
 - Configuring Project-specific Runners
 - Editing Project-specific Configuration
 - Triggering Build Retention in Artifactory
 - Scanning Builds with JFrog Xray
 - Running License Checks
 - Generic Build Integration
 - File Specs
 - Legacy Patterns (deprecated)
 - Attaching Searchable Parameters to Build-Info and to Published Artifacts
 - Black Duck Code Center Integration (deprecated)
 - Viewing Project-specific Configuration
- Running a Build with the Artifactory Plugin
- Triggering Builds in Reaction to Changes in Artifactory
- Proxy Configuration
- Licence
- Change Log

Read More

- [TeamCity Artifactory Plugin - Release Management](#)

Installing the Plugin

Plugins are deployed to TeamCity by placing the packaged plugin into the `$(TeamCity Data Directory)/plugins directory` and restarting TeamCity. You can also accomplish this via the TeamCity UI via **Administration | Plugins List | Upload Plugin Zip** and choosing the zip-file from your file-system. You will need to restart TeamCity (tomcat) for the plugin to take effect.

Download the latest version of the plugin:

Remove older versions

If you have an older version of the plug-in, be sure to remove it before upgrading to a newer one

Configuration

To use the TeamCity Artifactory plugin you first need to configure your Artifactory servers in TeamCity's server configuration. You can then set up a project build runner to deploy artifacts and Build Info to a repository on one of the Artifactory servers configured.

Configuring System-wide Artifactory Servers

To make Artifactory servers globally available to project runner configurations, they must be defined in **Administration | Integrations | Artifactory**.

Select **Create new Artifactory server configuration** and fill in the URL of the Artifactory server.

Deployer credentials can be set at the global level for all builds, but they can also be overridden and set at a project build level.

Specifying a username and password for the resolver repository is optional. It is only used when querying Artifactory's REST API for a list of configured repositories and then only if the target instance does not allow anonymous access.

The screenshot shows the TeamCity Administration interface. At the top, there are navigation tabs: Projects, Changes, Agents (1), and Build Queue (0). The user is logged in as 'admin'. The main navigation menu on the left includes Project-related Settings (Projects, Build History Clean-up, Disk Usage, Server Health, Audit), User Management (Users, Groups, Roles), Integrations (Issue Tracker, NuGet Settings, Tools, Artifactory), and Server Administration (Global Settings). The 'Artifactory' integration is selected. A yellow banner at the top of the main content area states 'Artifactory server configuration was updated.' Below this is a table titled 'Artifactory Server URL' with columns for the URL and Actions (edit, delete). A modal dialog titled 'Edit Artifactory Server Configuration' is open, showing the following fields: Artifactory Server URL (http://localhost:8081/artifactory), Default Deployer Username (admin), Default Deployer Password (masked), Use Different Resolver Credentials (checked), Default Resolver Username (reader), Default Resolver Password (masked), and Timeout (300). Buttons for 'Cancel', 'Save', and 'Test connection' are at the bottom of the dialog.

Configuring Project-specific Runners

Editing Project-specific Configuration

To set up a project runner to deploy build info and artifacts to Artifactory go to **Administration | Projects** and select the project you want to configure.

Then, under the **Build Configurations** section, click the **Edit** link for the build you want to configure.

Under **Build Configuration Settings**, select the relevant **Build Step** and click the **Edit** link for the build step you want to configure.

When you select a value in the **Artifactory server URL** field, the selected server is queried for a list of configured repositories (using the credentials configured in the corresponding **Artifactory Server Configuration**). This populates the **Target Repository** field with a list of repositories to which you can select to deploy.

Clicking on the **Free-text mode** checkbox enables you to type in repository name as free text. You may also include variables as part of the text. For example: `libs-%variableName%`

Configuration errors

If the **Target Repository** list remains empty, check that the specified Artifactory server URL, credentials and proxy information (if provided) are valid.

Any information about communication errors that might occur can be found in the TeamCity server logs.

Deploy Artifacts To Artifactory	
Artifactory server URL:	<input type="text" value="http://localhost:8081/artifactory"/> Select an Artifactory server.
Target repository:	<input type="text" value="libs-release-local"/> <input type="checkbox"/> Free-text mode Specify a target deployment repository.
Target snapshot repository:	<input type="text" value="libs-snapshot-local"/> <input type="checkbox"/> Free-text mode Specify a target deployment.
Override default deployer credentials:	<input checked="" type="checkbox"/> Use different deployer user name and password than the default ones defined in the Artifactory settings under the administrative system configuration page. If no global defaults are defined and no user name and password are given here anonymous deployment will be attempted.
Deployer username:	<input type="text" value="admin"/> Name of a user with deployment permissions on the target repository.
Deployer password:	<input type="password" value="....."/> The password of the user entered above.
Deploy Maven artifacts:	<input checked="" type="checkbox"/> Uncheck if you do not wish to deploy Maven artifacts from the plugin (a more efficient alternative to Mavens own deploy goal).
Deployment include patterns:	<input type="text"/> Comma or space-separated list of Ant-style patterns of files that will be included in publishing. Include patterns are applied on the published file path before any exclude patterns.
Deployment exclude patterns:	<input type="text"/> Comma or space-separated list of Ant-style patterns of files that will be excluded from publishing. Exclude patterns are applied on the published file path after any include patterns.
Publish build info:	<input checked="" type="checkbox"/> Uncheck if you do not wish to deploy build information from the plugin.
Include Environment Variables:	<input type="checkbox"/> Check if you wish to include all environment variables accessible by the builds process.
Run license checks:	<input type="checkbox"/> Check if you wish automatic license scanning to run after the build is complete. (Requires Artifactory Pro).

Triggering Build Retention in Artifactory

You can trigger build retention when publishing build-info to Artifactory.

Discard Old Builds:	<input checked="" type="checkbox"/> Check if you wish to include discard old builds.
Days To Keep Builds:	<input type="text"/> If not empty, builds are only kept up to this number of days.
Max # Of Builds To Keep:	<input type="text"/> If not empty, only up to this number of builds are kept.
Exclude Builds:	<input type="text"/> Comma or space-separated list of build numbers to be exclude during the retention procedure.
Delete Artifact:	<input type="checkbox"/> Check for deleting artifacts during the build retention procedure.
Async Build Retention:	<input type="checkbox"/> Check for asynchronous build retention.

Scanning Builds with JFrog Xray

The TeamCity Artifactory Plugin is integrated with JFrog Xray through JFrog Artifactory, allowing you to have build artifacts and dependencies scanned for vulnerabilities and other issues. If issues or vulnerabilities are found, you may choose to fail a build job. The scan result details are always printed into the build log. This integration requires **JFrog Artifactory v4.16** and above and **JFrog Xray v1.6** and above.

Run Xray scan on build:	<input checked="" type="checkbox"/> Check if you wish to scan the build for vulnerabilities (Requires Artifactory Pro with Jfrog Xray).
Fail build if found vulnerable:	<input checked="" type="checkbox"/> Uncheck if you do not wish to fail the build if found vulnerable.

Running License Checks

If you are using Artifactory Pro, you can benefit from the [License Control](#) feature to discover and handle third party dependency licensing issues as part of the build.

If you check the **Run License Checks** checkbox, Artifactory will scan and check the licenses of all dependencies used by this build. You can also specify a list of recipients who should receive any license violation notifications by email.

Generic Build Integration

Generic build integration provides Build Info support for the following runner types:

- Command Line
- FxCop
- MSBuild
- Rake
- Powershell
- XCode Project
- NuGet Publish
- NAnt
- Visual Studio (sln)
- Visual Studio 2003
- SBT, Scala build tool

This allows the above builds to:

1. Upload any artifacts to Artifactory, together with custom properties metadata, and keep published artifacts associated with the TeamCity build.
2. Download artifacts from Artifactory that are required by your build.

You can define the artifacts to upload and download by either using "File Specs" or "Legacy Patterns".

File Specs

File Spec are specified in JSON format. You can read the File Spec schema [here](#).

Legacy Patterns (deprecated)

Legacy patterns are deprecated since version 1.8.0 and will be removed in future releases.

Custom published artifacts	Allows you to specify which artifact files produced by the build should be published to Artifactory. At the end of the build the plugin locates artifacts in the build's checkout directory according to the specified artifact patterns, and publishes them to Artifactory to one or more locations, optionally applying a mapping for the target path of each deployed artifact. The pattern and mapping syntax for Published Artifacts is similar to the one used by TeamCity for Build Artifacts .
Custom build dependencies	Allows you specify dependency patterns for published artifacts that should be downloaded from Artifactory before the build is run. You can have detailed control over which artifacts are resolved and downloaded by using query-based resolution, adding to your artifact paths a query with the properties that the artifact should have before it can be downloaded. For further information read here about Resolution by Properties .

Custom published artifacts:

Edit published artifacts:

```
D:\Work\AntExample2\**\*.class=>class2.zip
D:\Work\AntExample2\**\*.class=>class2
```

New line or comma separated paths to build artifacts that will be published to Artifactory. Supports ant-style wildcards like dir/**/* .zip and target directories like *.zip=>winFiles, unix/distro.tgz=>linuxFiles, where winFiles and linuxFiles are target directories. Artifacts archiving is also supported, for example: test-results=>test-results.zip, where test-results is a source directory.

Custom build dependencies:

Edit build dependencies:

```
**/*.jar@multi :: multi#*=> one-jar-build
**/*.class@GENERIC :: ant#22=>generic-class
**/*.jar@gradle-perforce :: GRADLE#LATEST
```

New line or comma separated references to other build artifacts that this build should use as dependencies. Each reference is specified in the format of: repo_key:path_pattern[:prop=val1,val2[:prop2+=val3]]@build_name#build_number[=>target_dir], where: repo_key - A key of the Artifactory repository that contains the dependencies (may contain the * and the ? wildcards). path_pattern - An Ant-like pattern of the dependencies path within the Artifactory (may contain the * and the ? wildcards, including **). For example: repo-key:dir/**/bob/*.zip (** wildcards are not supported) Artifacts can be downloaded conditionally based on their property values in Artifactory. For example, to download all zip files marked as production ready: repo-key:dir/**/bob/*.zip:status==prod. For more details see the plug-in's [documentation](#). build_name - The name of the build that was published to Artifactory, of which dependencies should be resolved. build_number - A specific build run number. Can be LATEST to depend on the latest build run, or LAST_RELEASE to depend on the latest build with a "release" status. For example: repo-key:dir/**/bob/*.zip@myBuild#LATEST target_dir - An optional target directory to where resolved dependencies will be downloaded. By default dependencies will be downloaded to a path under the build workspace. For example: repo-key:*.zip=>winFiles, repo-key:unix/distro.tgz=>linuxFiles, where winFiles and linuxFiles are target directories. Target directories can either be absolute or relative to the working directory.

As of version 2.1.4, the above configuration is not backward compatible and you may need to re-save the builds configuration for them to run properly.

If no matching artifacts are found, remember that these parameters may be case sensitive depending on the operating system, the agent and the server they are running on.

Attaching Searchable Parameters to Build-Info and to Published Artifacts

In the **Build Configuration Settings** you can select **Parameters** to define system properties or environment variables that should be attached to artifacts and their corresponding build info.

To define a parameter click on the **Add new parameter** button.

Add New Parameter ✕

Name:

Kind: Configuration parameter

Value: 📄
Type '%' for reference completion

Spec: Edit...
[Show raw value](#)
Defines parameter control presentation and validation.

Save Cancel

Fill in the corresponding fields.

Parameters relevant for builds run through Artifactory are:

- buildInfo.property.* - All properties starting with this prefix are added to the root properties of the build-info
- artifactory.deploy.* - All properties starting with this prefix are attached to any deployed produced artifacts

You can specify all the properties in a single file, and then define another property pointing to it.

To point the plugin to a properties file, define a property called `buildInfoConfig.propertiesFile` and set its value to the absolute path of the properties file.

It is also possible to point the plugin to a properties file containing the aforementioned properties.

The properties file should be located on the machine running the build agent, **not on the server!**

[+ Add new parameter](#)

Configuration Parameters

Configuration parameters are not passed into build, can be used in references only. [?](#)

None defined

System Properties (system.)

System properties will be passed into the build (without system. prefix), they are only supported by the build runners that understand the property notion. [?](#)

Name	Value		
system.buildinfo.property.bob.number	%system.build.vcs.number%	Edit	Delete
system.buildInfoConfig.propertiesFile	this/is/a/path.properties	Edit	Delete

Environment Variables (env.)

Environment variables will be added to the environment of the processes launched by the build runner (without env. prefix). [?](#)

Name	Value		
env.buildInfoConfig.deploy.foo	%maven.project.name%	Edit	Delete

Black Duck Code Center Integration (deprecated)

This feature is no longer supported since version 5 of Artifactory.

If you are using Artifactory Pro and have an account with [Black Duck Code Center](#), you can run the build through an automated, non-invasive, open source component approval process, and monitor for security vulnerabilities.

Run Black Duck Code Center compliance checks:
Check if you wish that automatic Black Duck Code Center compliance checks will occur after the build is completed. (Requires Artifactory Pro).

Code Center application name: [?](#)
The existing Black Duck Code Center application name.

Code Center application version: [?](#)
The existing Black Duck Code Center application version.

Send compliance report email to: [?](#)
Input recipients that will receive a report email after the automatic Black Duck Code Center compliance checks finished.

Limit checks to the following scopes: [?](#)
A list of dependency scopes/configurations to run Black Duck Code Center compliance checks on. If left empty all dependencies from all scopes will be checked.

Include published artifacts:
Include the build's published module artifacts in the Black Duck Code Center compliance checks if they are also used as dependencies for other modules in this build.

Auto-create missing component requests:
Auto create missing components in Black Duck Code Center application after the build is completed and deployed in Artifactory.

Auto-discard stale component requests:
Auto discard stale components in Black Duck Code Center application after the build is completed and deployed in Artifactory.

Viewing Project-specific Configuration

Existing project configuration can be viewed in **Settings** under **Projects | \$PROJECT_NAME | \$BUILD_NAME**:

Build Step: Maven [edit »](#)

Step 1:

Runner type: **Maven** (Runs Maven builds)
Execute: ⓘ **If all previous steps finished successfully (zero exit code)**
POM file path: **maven-example/pom.xml**
Goals: **clean install**
Maven used: default
Additional Maven command line parameters: **none specified**
User settings provided by default
Maven metadata disabled: **false**
Use own local artifact repository: **false**
Build only modules affected by changes (incremental building): **false**
JDK home path: **not specified**
Build working directory: **not specified**
JVM command line parameters: **not specified**

Deploy artifacts to Artifactory: **enabled**
Artifactory server: **http://10.0.0.235:8080/artifactory**
Target repository: **libs-release-local**
Target snapshot repository: **libs-snapshot-local**
Deployer username: **admin**
Deploy artifacts: **true**
Deployment include patterns: **not specified**
Deployment exclude patterns: **not specified**
Publish Build Info: **true**
Include Environment Variables: **true**
Environment Variables Include Patterns: **not specified**
Environment Variables Exclude Patterns: ***password*,*secret***
Run license checks (Requires pro): **true**
License violation notifications recipients: **none specified**
Limit checks to the following scopes: **none specified**
Include published artifacts: **false**
Disable automatic license discovery: **false**
Enable Artifactory release management: **true**
VCS tags base URL/name: **none**
Git release branch name prefix: **REL-BRANCH-**
Alternative Maven goals: **none**
Alternative Maven command line parameters: **none**
Default module version configuration: **Global**

Java code coverage: **disabled**

Running a Build with the Artifactory Plugin

Once you have completed setting up a project runner you can run a project build. The Artifactory plugin takes effect at the end of the build and does the following:

1. For all build runner types - Publishes the specified Published Artifacts to the selected target repository and applies corresponding path mappings.
2. For Maven or Ivy/Ant build runner - Deploys all artifacts to the selected target repository together at the end of the build (as opposed to deploying separately at the end of each module build as done by Maven and Ivy).
3. Deploys the Artifactory BuildInfo to Artifactory, providing [full traceability of the build in Artifactory](#), with links back to the build in TeamCity.

```

[15:38:09]: [INFO] [source:jar {execution: attach-sources}]
[15:38:09]: [INFO] Building jar: /home/noam/Work/JetBrains/tc-5.1/buildAgent/work/db3222d3a8dbdddf/build-info-extractor-gradle/target/build
[15:38:09]: [INFO] [install:install {execution: default-install}]
[15:38:09]: [INFO] Installing /home/noam/Work/JetBrains/tc-5.1/buildAgent/work/db3222d3a8dbdddf/build-info-extractor-gradle/target/build-in
[15:38:10]: [INFO] Installing /home/noam/Work/JetBrains/tc-5.1/buildAgent/work/db3222d3a8dbdddf/build-info-extractor-gradle/target/build-in
[15:38:10]: [INFO]
[15:38:10]: [INFO]
[15:38:10]: [INFO] Reactor Summary:
[15:38:10]: [INFO] -----
[15:38:10]: [INFO] JFrog Build-Info ..... SUCCESS [3.653s]
[15:38:10]: [INFO] JFrog Build-Info API ..... SUCCESS [6.799s]
[15:38:10]: [INFO] JFrog Build-Info Client ..... SUCCESS [3.941s]
[15:38:10]: [INFO] JFrog Build-Info Extractor ..... SUCCESS [2.566s]
[15:38:10]: [INFO] JFrog Build-Info Maven 3 Extractor ..... SUCCESS [3.415s]
[15:38:10]: [INFO] Build Info Gradle Extractor ..... SUCCESS [2.006s]
[15:38:10]: [INFO] -----
[15:38:10]: [INFO] BUILD SUCCESSFUL
[15:38:10]: [INFO] -----
[15:38:10]: [INFO] Total time: 22 seconds
[15:38:10]: [INFO] Finished at: Sun Apr 25 15:38:10 IDT 2010
[15:38:10]: [INFO] Final Memory: 78M/479M
[15:38:10]: [INFO] -----
[15:38:10]: [Maven Watcher] starting handling projects
[15:38:10]: [Maven Watcher] building report document...
[15:38:10]: [Maven Watcher] building report document done
[15:38:10]: [Maven Watcher] writing report to /home/noam/Work/JetBrains/tc-5.1/buildAgent/temp/buildTmp/maven-build-info.xml
[15:38:10]: [Maven Watcher] done writing report
[15:38:10]: Number of processed tests: 35
[15:38:11]: Deploying artifacts to http://amphibian.jfrog.org:8081/artifactory
[15:38:11]: Deploying artifact: org/jfrog/build-info-parent/1.2.x-SNAPSHOT/build-info-parent-1.2.x-SNAPSHOT.pom
[15:38:13]: Deploying artifact: org/jfrog/build-info-api/1.2.x-SNAPSHOT/build-info-api-1.2.x-SNAPSHOT.jar
[15:38:13]: Deploying artifact: org/jfrog/build-info-api/1.2.x-SNAPSHOT/build-info-api-1.2.x-SNAPSHOT-sources.jar
[15:38:13]: Deploying artifact: org/jfrog/build-info-client/1.2.x-SNAPSHOT/build-info-client-1.2.x-SNAPSHOT.jar
[15:38:13]: Deploying artifact: org/jfrog/build-info-client/1.2.x-SNAPSHOT/build-info-client-1.2.x-SNAPSHOT-sources.jar
[15:38:13]: Deploying artifact: org/jfrog/build-info-extractor/1.2.x-SNAPSHOT/build-info-extractor-1.2.x-SNAPSHOT.jar
[15:38:13]: Deploying artifact: org/jfrog/build-info-extractor/1.2.x-SNAPSHOT/build-info-extractor-1.2.x-SNAPSHOT-sources.jar
[15:38:13]: Deploying artifact: org/jfrog/build-info-extractor-maven3/1.2.x-SNAPSHOT/build-info-extractor-maven3-1.2.x-SNAPSHOT.jar
[15:38:13]: Deploying artifact: org/jfrog/build-info-extractor-maven3/1.2.x-SNAPSHOT/build-info-extractor-maven3-1.2.x-SNAPSHOT-sources.jar
[15:38:13]: Deploying artifact: org/jfrog/build-info-extractor-gradle/1.0-SNAPSHOT/build-info-extractor-gradle-1.0-SNAPSHOT.jar
[15:38:13]: Deploying artifact: org/jfrog/build-info-extractor-gradle/1.0-SNAPSHOT/build-info-extractor-gradle-1.0-SNAPSHOT-sources.jar
[15:38:13]: Deploying build info ...
[15:38:14]: Build finished

```

[? Help](#) [Feedback](#)

TeamCity Enterprise 5.1 (build 13360)

You can also link directly to the build information in Artifactory from a build run view:

The screenshot shows the TeamCity interface for a build run. At the top, there are navigation tabs: **Projects**, **Changes**, **Agents** (1), and **Build Queue** (0). The user is logged in as **admin**. The current view is for a build run: **Test > Maven > #9 (16 Apr 14 12:56)**. Below this, there are buttons for **Run**, **Actions**, and **Edit Configuration Settings**. A secondary navigation bar shows **Overview**, **Changes**, **Tests**, **Build Log**, **Parameters**, **Artifacts**, and **Maven Build Info**. The **Maven Build Info** tab is active, showing a green box with the following details: **Result: Tests passed: 2**, **Time: 16 Apr 14 12:56:47 - 12:56:56 (9s)**, **Branch: master**, **Agent: Default Agent**, and **Triggered by: you on 16 Apr 14 12:56**. Below this, there is a link for **Artifactory Build Info** and a status indicator: **2 tests passed**.

Triggering Builds in Reaction to Changes in Artifactory

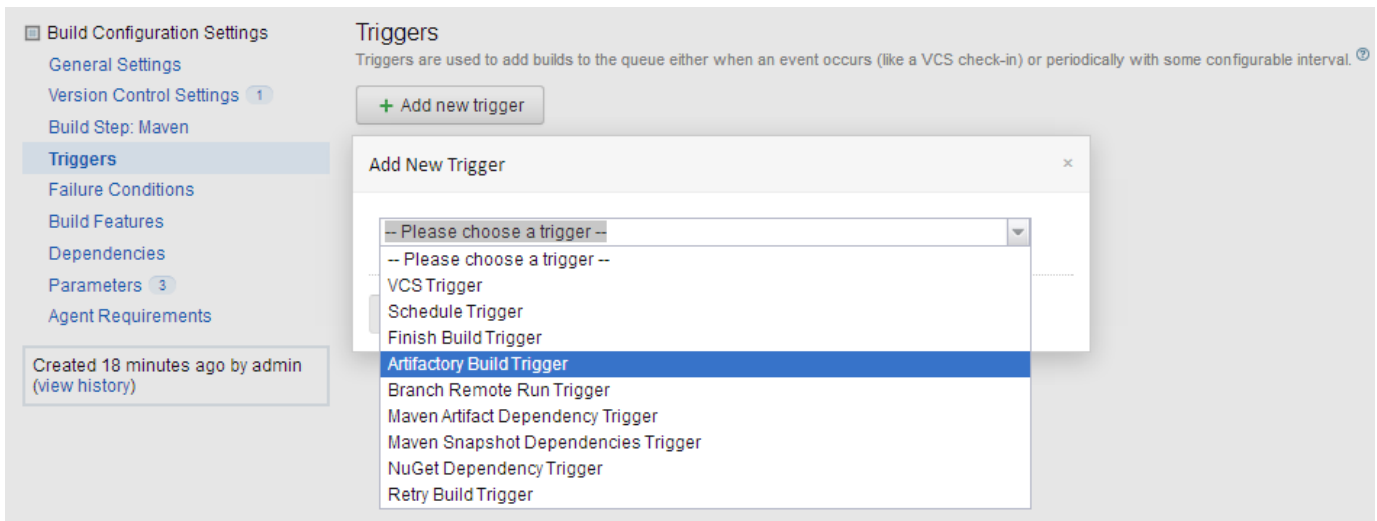
The plugin allows you to set a new type of trigger that periodically polls a path in Artifactory, a folder or an individual file. Whenever a change is detected in the polled element, the TeamCity build is triggered. For example, the build could be triggered when new artifacts have been deployed to the specified path in Artifactory.

Artifactory Pro required

Triggering builds is only available with Artifactory Pro

To configure a new build trigger, under **Administration**, select **\$PROJECT_NAME | \$BUILD_NAME**. Then, under **Build Configuration Settings** select **Triggers**.

Click the **Add new trigger** button to select an **Artifactory Build Trigger**



Select the **Artifactory Server URL** and the **Target repository**.

Complete the username and a password fields of a valid deployer for the selected repository.

Deploy permission

The specified user must have deploy permissions on the repository

Then, in **Items to watch**, specify the paths in the selected repository in which a change should automatically trigger a build.

Add New Trigger ✕

Artifactory Build Trigger

Artifactory Server Settings

Artifactory server URL: * ⓘ

Target repository: * ⓘ

Username: ⓘ

Password: ⓘ

Items to watch: * ⓘ

Edit items to watch:

```
com/acme/releases/milestone5
com/acme/snapshots
```

New line or comma separated paths in Artifactory that will be polled for changes. Paths denote files or folders relative to the target repository. For example:
`com/acme/releases/milestone-5.`

Note: Artifactory folders are scanned for changes recursively. To avoid slow performance it is recommended to narrow down the scope of scanning.

Polling interval seconds: ⓘ

Be as specific as possible in Items to watch

In order to establish if there has been a change, Artifactory must traverse all the folders and their sub-folders specified in **Items to watch**. If the specified folders have a lot of content and sub-folders, this is a resource intensive operation that can take a long time.

Therefore, we recommend being as specific as possible when specifying folders in **Items to watch**.

Proxy Configuration

If the Artifactory server is accessed via a proxy, you need to configure the proxy by setting the following properties in the `$TEAMCITY_USER_HOME/.BuildServer/config/internal.properties` file. If the file does not exist, you'll need to create it.

```
org.jfrog.artifactory.proxy.host
org.jfrog.artifactory.proxy.port
org.jfrog.artifactory.proxy.username
org.jfrog.artifactory.proxy.password
```

Since version 2.5.0, you can also define a proxy for specific build agents. You do that by adding the TeamCity agent name to the end of the above

property names.

For example, if you wish to configure a proxy for the "my-agent" agent, the proxy properties configuration should look as follows:

```
org.jfrog.artifactory.proxy.host.my-agent
org.jfrog.artifactory.proxy.port.my-agent
org.jfrog.artifactory.proxy.username.my-agent
org.jfrog.artifactory.proxy.password.my-agent
```

In case your build agent name contains a white-space, you should replace the white-space in the property name with `\u0020`.

For example, here's how you define the proxy host for the "Default Agent":

```
org.jfrog.artifactory.proxy.host.Default\u0020Agent
```

Licence

The TeamCity Artifactory plugin is available under the Apache v2 License.

Change Log

[Click to see change log details...](#)

2.5.0 (28 Sep 2017)

1. Allow proxy configuration per agent (TCAP-237)
2. Support pattern exclusion in File Specs (TCAP-300)
3. File specs AQL optimizations (TCAP-302)
4. Bug fixes (TCAP-297, TCAP-299, TCAP-301, TCAP-303)

2.4.0 (29 Jun 2017)

1. Support for retention policy within TeamCity Plugin (TCAP-283)
2. Support Xray build scan (TCAP-292)
3. Add upload and download from file specs support to generic jobs (TCAP-294)
4. Allow to extract supported formats using file specs (TCAP-295)
5. Bug fixes (TCAP-167, TCAP-293, TCAP-296)

2.3.1 (23 Jan 2017)

1. Support full path in specs (TCAP-287)
2. Add to file spec the ability to download artifact by build name and number (TCAP-288)
3. Change file Specs pattern (TCAP-285)

2.3.0 (13 Nov 2016)

1. Upload and download File Specs support to generic jobs (TCAP-284)

2.2.1 (19 May 2016)

1. Bug fix (TCAP-214)

2.2.0 (21 March 2016)

1. Bug fixes (TCAP-238, TCAP-239, TCAP-241, TCAP-244, TCAP-245, TCAP-247, TCAP-250, TCAP-258, TCAP-236)

2.1.13 (4 May 2015)

1. Support multi Artifactory Build Triggers (TCAP-222)
2. Support SBT build tool (TCAP-223)
3. Bug fix (TCAP-214)

2.1.12 (12 Mar 2015)

1. Adding support for free text repository configuration ([TCAP-217](#))

2.1.11 (7 Dec 2014)

1. Compatibility with Gradle 2.x ([TCAP-211](#))
2. Bug Fixed ([TCAP-205](#))

2.1.10 (8 May 2014)

1. Bug Fixed ([TCAP-206](#), [TCAP-72](#))

2.1.9 (17 Apr 2014)

1. Adding Version Control Url property to the Artifactory Build Info JSON. ([TCAP-203](#))
2. Support for TeamCity 8.1 Release management feature issues
3. Support working with maven 3.1.1
4. Bug Fixed ([TCAP-197](#), [TCAP-161](#))

2.1.8 (15 Jan 2014)

1. Allow remote repository caches to be used for build triggering - [TCAP-196](#)
2. Bug Fixes

2.1.7 (18 Dec 2013)

1. Add support for blackduck integration - [TCAP-185](#)

2.1.6 (03 Sep 2013)

1. TeamCity 8.0.x full compatability issue - [TCAP-172](#)
2. Global and build credentials issue - [TCAP-153](#)
3. Repositories refreshed by credential issue - [TCAP-166](#)
4. Generic deployment resolution on Xcode builds - [TCAP-180](#)
5. Working directory in Gradle build issue - [TCAP-125](#)

2.1.5 (07 Jul 2013)

1. Fix security issue - [TCAP-172](#)
2. Improve generic resolution - [BI-152](#)

2.1.4 (21 Aug 2012)

1. Compatible with TeamCity7.1.
2. Bug Fixes

2.1.3 (30 May 2012)

1. Compatible with TeamCity7.
2. Support 'Perforce' in release management.
3. Support multiple deploy targets for the same source pattern in generic deploy.
4. Support for custom build dependencies resolution per build.

2.1.2 (12 Dec 2011)

1. Compatible with Gradle 1.0-milestone-6.

2.1.1 (09 Aug 2011)

1. Support for Gradle milestone-4
2. Better support for releasing nested Maven projects
3. Fixed minor Maven deployments discrepancies

2.1.0 (14 Jul 2011)

1. Release management capabilities
2. Bug fixes

2.0.1 (9 Jan 2011)

1. Auto Snapshot/Release target repository selection
2. Add ivy/artifact deploy patterns
3. Improved Gradle support
4. Bug fixes

2.0.0 (5 Dec 2010)

1. Support for Gradle builds
2. Support for maven3 builds
3. Default deployer add resolver credentials
4. Support for multi steps builds

1.1.3 (21 Nov 2010)

1. Include/exclude pattern for artifacts deployment

1.1.2 (7 Nov 2010)

1. Control for including published artifacts when running license checks
2. Limiting license checks to scopes
3. Control for turning off license discovery when running license checks

TeamCity Artifactory Plugin - Release Management

Overview

The TeamCity Artifactory Plugin includes release management capabilities for Maven and Gradle runners that use Subversion, Git or Perforce for version control.

When you run your builds using [Maven](#) or [Gradle](#) with jobs that use [Subversion](#), [Git](#) or [Perforce](#) as your version control system, you can manually stage a release build allowing you to:

- Change values for the release and next development version
- Choose a target staging repository to which to deploy the release
- Create a VCS tag for the release

Staged release builds can later be promoted or rolled-back, changing their release status in Artifactory, and build artifacts may optionally be moved to a different target repository.

Inside Artifactory, the history of all build status changes (staged, promoted, rolled-back, etc.) is recorded and displayed for full traceability.

Page Contents

- [Overview](#)
- [Before Getting Started](#)
 - [Working with Git](#)
 - [Working with Subversion](#)
- [Maven Release Management](#)
 - [Configuring Maven Runners](#)
 - [Staging a Maven Release Build](#)
 - [Promoting a Release Build](#)
- [Gradle Release Management](#)
 - [Configuring Gradle Runners](#)
 - [Staging a Gradle Release Build](#)
 - [Promoting a Release Build](#)

Before Getting Started

Working with Git

Pre-requisites for using Release Management with Git:

1. Since the Release Management process runs git commands, the git client must be installed on the TeamCity build agent, using an ssh key (using the git client with user and password is not supported)..
2. The git client should be configured with an ssh key, so that it can access your Git repositories. Therefore, before running the Release Management process for the first time, it is recommended that you first make sure that you're able to perform a git push from the build agent console. Also, make sure that the git push command runs without displaying a user prompt. **Note that configuring an ssh passphrase for the git client is not supported.**

Changes are only committed if the files are modified (POM files or *gradle.properties*)

During the release, the plugin performs the following steps:

1. If **Create Branch** is checked, create and switch to the release branch.
2. Commit the release version to the current branch.
3. Create a release tag.
4. Push the changes.
5. Switch to the checkout branch and commit the next development version.
6. Push the next development version to the working branch.

Working with Subversion

Release management with [TeamCity Artifactory Plug-in](#) supports Subversion when using one checkout directory.

During the release the plugin does the following:

1. Commits the release version directly to the tag (if **Create tag** is checked). The release version is not committed to the working branch.
2. Commits the next development version to the working branch.

Maven Release Management

The [TeamCity Artifactory Plugin](#) manages a release with Maven running the build only once using the following basic steps:

1. Change the POM versions to the release version (before the build starts).
2. Trigger the Maven build (with optionally different goals).
3. Commit/push changes to the tag (Subversion) or the release branch (Git).
4. Change the POM versions to the next development version.
5. Commit/push changes to the trunk.

If the build fails, the plugin attempts to rollback the changes (both local and committed).

For more information including configuration of Maven Runners, and Jobs and staging a release build, please refer to [TeamCity Artifactory Plugin](#).

Configuring Maven Runners

To enable release management in Maven runners, edit the runner's step configuration and check the **Enable Artifactory release management** checkbox.

Enable Artifactory release management:



VCS tags base URL/name:

For subversion this is the URL of the tags location, for Git this is the name of the tag.

Git release branch name prefix:

The prefix of the release branch name (applicable only to Git).

Alternative Maven goals:

Alternative goals to execute for a Maven build running as part of the release. If left empty, the build will use original goals instead of replacing them.

Alternative Maven command line parameters:

Alternative options to execute for a Maven build running as part of the release. If left empty, the build will use original options instead of replacing them.

Default module version configuration:

Default versioning strategy to use:

- Global - One version for all modules
- Per Module - Allows different version per module
- None - Don't change module version

Staging a Maven Release Build

Once release management is enabled, the Artifactory Release Management tab appears at the top of the build page.

Run ...

Settings Maven 2 Artifactory Release Management

Clicking on the tab reveals configuration options for the release build:

Artifactory Release Staging

Last build version: 2.14-SNAPSHOT

Module Version Configuration

One version for all modules

Use same version for all modules.

Version per module

Select version per module.

Use existing module versions

Don't change module versions.

VCS Configuration

Create VCS tag

Create tag in the version control system.

Tag URL/name:

The tag URL. Build will fail if the tag already exists.

Tag comment:

The comment to use when creating the tag.

Next development version comment:

The comment to use when committing changes for the next development version.

Artifactory Configuration

Repository to stage the release to (Artifactory Pro):

Target repository to stage the release published artifacts to.

Staging comment:

Comment that will be added to the promotion action.

Build and Release to Artifactory

The release staging page displays the last version built (the version tag is that of the root POM, and is taken from the last build that is not a release). Most of the fields in the form are populated with default values.

Version configuration controls how the plugin changes the version in the POM files (global version for all modules, version per module or no version changes).

If the **Create VCS tag** checkbox is checked (default), the plugin commits/pushes the POMs with the release version to the version control system with the commit comment. When using Git, there's also an option to create a release branch.

Click on the **Build and Release to Artifactory** button to trigger the release build.

Target server is Artifactory Pro?

If the target Artifactory server is a Pro edition, you can change the target repository, (the default is the release repository configured in Artifactory publisher) and add a staging comment which is included in the build info deployed to Artifactory.

Promoting a Release Build

You can promote a release build after it completes successfully.

This is not a mandatory step but is very useful because it allows you to mark the build as released in Artifactory, and move or copy the build

artifacts to another repository so they are available to other users.

To promote a build, browse to the build's result page and click the **Artifactory Release Promotion** link.

Artifactory Pro Required

Promotion features are only available with Artifactory Pro

- Overview
- Changes (0)
- Tests
- Build Log
- Build Parameters
- Maven Build Info

Result: ● Tests passed: 2

Time: 06 Jul 11 14:47:28 - 14:47:57 (29s)



Artifactory Build Info



Artifactory Release Promotion



2 tests passed

Clicking on the link will open the **Release Promotion** dialog:

Artifactory Pro Release Promotion

Target status: * ? Released

Target promotion repository: ? libs-release-local

Comment: ?

Include dependencies: ?

Use Copy: ?

Promote Close

Select the target status of the build ("Released" or "Rolled-Back"). You may also enter a comment to display in the build in Artifactory.

To move or copy the build artifacts, select the **Target promotion repository**.

Gradle Release Management

The **TeamCity Artifactory Plugin** supports release management when running builds with Gradle. This relies on the version property (and others) managed by the `gradle.properties` file. The plugin reads the properties from the Artifactory release management configuration, and modifies those properties in the `gradle.properties` file.

The plugin manages a release using the following basic steps:

1. Modify properties in the `gradle.properties` to release values (before the build starts).
2. Trigger the Gradle build (with optionally different tasks and options).
3. Commit/push changes to the tag (Subversion) or the release branch (Git)

4. Modify the `gradle.properties` to the next integration values.
5. Commit/push changes to the trunk.

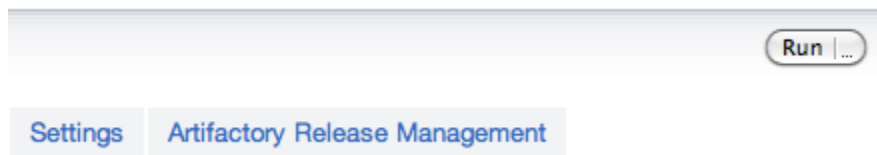
Configuring Gradle Runners

To enable release management for Gradle runners, edit the runner's step configuration and check the **Enable Artifactory release management** checkbox.

Enable Artifactory release management:	<input checked="" type="checkbox"/>
VCS tags base URL/name:	<input type="text" value="thetag"/> <small>For subversion this is the URL of the tags location, for Git this is the name of the tag.</small>
Git release branch name prefix:	<input type="text" value="REL-BRANCH-"/> <small>The prefix of the release branch name (applicable only to Git).</small>
Release properties:	<input type="text" value="latest-release-version,version"/> <small>Properties in your project's 'gradle.properties' file whose value should change upon release.</small>
Next integration properties:	<input type="text" value="other-version"/> <small>Properties in your project's 'gradle.properties' file whose value should change upon release, but also for work on the next integration/development version after the release has been created.</small>
Alternative Gradle tasks:	<input type="text" value="clean build"/> <small>Alternative tasks to execute for a Gradle build running as part of the release. If left empty, the build will use original tasks instead of replacing them.</small>
Alternative Gradle options:	<input type="text" value="-I init.gradle"/> <small>Alternative options to execute for a Gradle build running as part of the release. If left empty, the build will use original options instead of replacing them.</small>

Staging a Gradle Release Build

Once release management is enabled, the **Artifactory Release Management** tab appears at the top of the build page.



Clicking on the tab reveals configuration options for the release build:

build-info-extractor-maven3-version

Release value:

The release version to use in the `gradle.properties` and for tagging.

Next integration value:

The next development value to use in the `gradle.properties`.

VCS Configuration

Use release branch

Flag whether to create a Git release branch.

Release branch:

The name of the Git release branch.

Create VCS tag

Create tag in the version control system.

Tag URL/name:

The tag URL. Build will fail if the tag already exists.

Tag comment:

The comment to use when creating the tag.

Next development version comment:

The comment to use when committing changes for the next development version.

Artifactory Configuration

Repository to stage the release to (Artifactory Pro):

Target repository to stage the release published artifacts to.

Staging comment:

Comment that will be added to the promotion action.

The **Release staging** tab displays the **Release** and **Next development** properties configured for the runner. The plugin reads these values from the `gradle.properties` file and attempts to calculate and display **Release** and **Next integration version** in the text fields.

If **Create VCS tag** is checked (default), the plugin commits/pushes the POMs with the release version to the version control system with the commit comment. When using Git, if **Use release branch** is checked, the **Next release version** changes are carried out on the release branch instead of the current checkout branch. The final section allows you to change the target repository (the default is the release repository configured in Artifactory publisher) and an optional staging comment which includes the build info deployed to Artifactory.

Click on the **Build and Release to Artifactory** button to trigger the release build.

Promoting a Release Build

Promotion is the same as in [Promoting a Release Build for Maven](#).

Bamboo Artifactory Plug-in

Overview

Artifactory provides tight integration with Bamboo through the Bamboo Artifactory Plug-in. Beyond managing efficient deployment of your artifacts to Artifactory, the plug-in lets you capture information about artifacts deployed, dependencies resolved, environment data associated with the Bamboo build runs and more, that effectively facilitates fully traceable builds.

Build Runner Support

The Bamboo Artifactory Plug-in currently provides full support for **Maven 3**, **Gradle** and **Ivy** builds. **Generic Deployment Tasks** are available for all builder types.

Before you begin

Please refer to the general information about [Artifactory's Build Integration](#) before using the Bamboo Artifactory Plug-in.

Source Code Available!

The Bamboo Artifactory Plugin is [an open-source project on GitHub](#) which you can freely browse and fork.

Download

Latest

Download version 2.3.0 which is compatible with Bamboo 6.2.x and 6.1.x.

Upgrading to version 2.x from version 1.x of the plugin requires new installation steps. Please refer to [Installing the Plugin](#) for more details.

Older

Version	Download link	Compatibility
2.1.1	Download	Bamboo 6.0.x
2.1.0	Download	Bamboo 5.14.x
1.13.0	Download	Bamboo 5.14.x
1.11.2	Download	Bamboo 5.13.x
1.11.1	Download	Bamboo 5.12.x
1.10.3	Download	Bamboo 5.11.x
1.10.1	Download	Bamboo 5.10.x
1.9.2	Download	Bamboo 5.9.x
1.7.7	Download	Bamboo 5.8.x

Installing the Plugin

Requirements

- Artifactory 2.2.5 or later. For best results and optimized communication, we recommend using the latest version of Artifactory.
- [Artifactory Pro](#) is required for advanced features, such as [License Control](#) and enhanced [Build Integration](#).
- Maven 3.
- Gradle 0.9 or later.
- Ant and Ivy 2.1.0 or later.

Upgrading to Versions 2.x from Versions 1.x

If you are currently using a version of the plugin below 2.0.0 and would like to upgrade to version 2.0.0 or above, you need to migrate your Artifactory configuration data to the format expected by the type 2 plugin as described in the following steps:

1. If you are not already on version 1.13.0 of the plugin, upgrade to that version first.
2. From **Bamboo Administration | Artifactory Plugin**, click on the "*Migrate data to v2*" button.
3. Remove plugin version 1.13.0 and restart Bamboo.

4. You're now ready to install version 2.x according to the below instructions.

Installing Versions 2.x

From version 2.0.0, the Bamboo Artifactory Plugin is released as a type 2 plugin. You can read about installing type 2 plugins in the Bamboo documentation for [Installing add-ons](#).

Installing Versions 1.x

Remove older versions

If you have an older version of the plug-in, be sure to remove it before upgrading to a newer one

For versions below 2.0.0, the plugin was released as a type 1 plugin and is deployed to Bamboo by placing the packaged plugin jar file into the `$BAMBOO_INSTALLATION_HOME/atlassian-bamboo/WEB-INF/lib` folder and restarting Bamboo.

For more details please refer to the Bamboo documentation for [Installing Plugins type 1 add-ons](#).

Page Contents

- [Overview](#)
 - [Build Runner Support](#)
- [Download](#)
 - [Latest](#)
 - [Older](#)
- [Installing the Plugin](#)
 - [Requirements](#)
 - [Upgrading to Versions 2.x from Versions 1.x](#)
 - [Installing Versions 2.x](#)
 - [Installing Versions 1.x](#)
- [Configuration](#)
 - [Configuring Maven 3, Gradle and Ivy Builders](#)
 - [Configuring System-wide Artifactory Server\(s\)](#)
 - [Configuring a Project Builder](#)
 - [Selecting Artifactory Servers and Repositories](#)
 - [Running Licence Checks](#)
 - [Black Duck Code Center Integration](#)
 - [The Artifactory Generic Resolve Task](#)
 - [The Artifactory Generic Deploy Task](#)
 - [Using File Specs](#)
 - [The Artifactory Deployment Task](#)
- [License](#)
 - [Attaching Searchable Parameters](#)
 - [Overriding Plan values using Bamboo Variables](#)
 - [Release Management](#)
 - [Push to Bintray](#)
- [Changelog](#)

Read More

- [Bamboo Artifactory Plugin - Release Management](#)

Configuration

To use the Bamboo Artifactory plug-in you need to set up your Artifactory server(s) in Bamboo's server configuration. You can then set up a project builder to deploy artifacts and build-info to a repository on one of the configured Artifactory servers.

Configuring Maven 3, Gradle and Ivy Builders

Before you begin with any Artifactory-specific setup, ensure that Maven 3, Gradle and/or Ivy builders are available for project configurations.

These builders are defined as **Server Capabilities** in Bamboo

To define Server Capabilities for builders:

1. Under the **Administration** menu, select **Overview** to view the **Bamboo administration** page.
2. Then, under **Build Resources** select **Server Capabilities**
3. Select **Executable** as the **Capability Type**

4. Select **Artifactory Maven 3**, **Artifactory Gradle** or **Artifactory Ivy** as the type from the **Types** list.
5. Make sure that **Path** points to an installation directory of the selected builder type.

Add capability

Capability type	Executable ▼
Type	Ant ▼
Executable label	Ant Artifactory Gradle Artifactory Ivy Artifactory Maven 3
Path	Command Grails Maven 1.x Maven 2.x Maven 3.x MSBuild NAnt Node.js JUnit Runner PHPUnit PHPUnit 3.3.x Visual Studio

Configuring System-wide Artifactory Server(s)

To make Artifactory servers available to project configurations, they must be defined under **Administration | Plugins | Artifactory Plugin**.

Enter the Artifactory server URL in the **Add New Server Configuration** fields.

Username and Password

Username and password are optional and are only used when querying Artifactory's REST API for a list of configured repositories (credentials are only required if the target instance does not allow anonymous access).

Manage Artifactory Servers

You can use this page to add, edit and delete Artifactory server configurations.

Artifactory Server URL	Username	Timeout	Operations
http://[redacted]:8081/artifactory	admin	300	Edit Delete

Add New Server Configuration

Artifactory Server Details

Configure an Artifactory server that will be available to project configurations for deployment of artifacts and build info. If anonymous user is enabled in Artifactory server, you can leave the username/password empty

Artifactory Server*
URL
Specify the root URL of your Artifactory installation, like `http://repo.jfrog.org/artifactory`

Username
User with permissions to query the list of Artifactory repositories. Leave empty if anonymous is enabled.

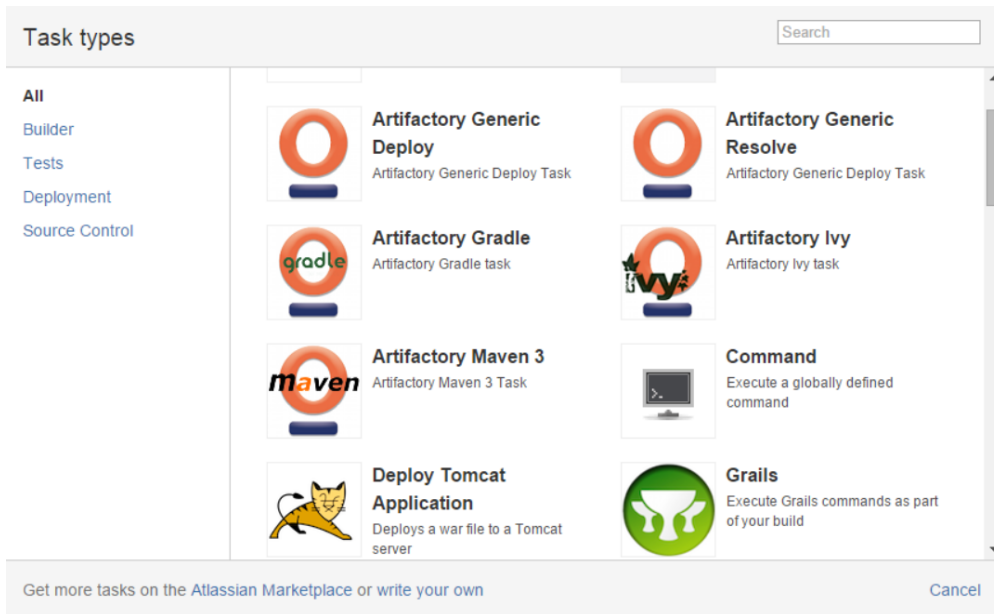
Password
Password of the user with permissions to query the list of Artifactory repositories. Leave empty if anonymous is enabled.

Request Timeout*
Network timeout in seconds to use both for connection establishment and for unanswered requests.

Configuring a Project Builder

To set up a project task to deploy build-info and artifacts to Artifactory:

1. Go to the **Tasks** step of your jobs configuration.
2. When adding a task type, select the Artifactory Maven 3, Gradle or Ivy builder.



3. The builder configuration fields appear and include Artifactory and build-info configuration options.

Selecting Artifactory Servers and Repositories

Select an Artifactory server URL for resolving artifacts and to deploy build artifacts in the corresponding fields.

If you have configured the [System Wide Artifactory Servers](#) correctly with the required credentials, then once you select an Artifactory server, the corresponding fields are populated to let you choose a **Resolution Repository** and **Target Repository**.

Repository list empty?

If the **Resolution Repository** or **Target Repository** fields remain empty, check that you have entered valid credentials when defining the Artifactory servers.

The **Target Repository** field is populated with a list of available target repositories as returned by the server (queried with the credentials in the server configuration, if provided).

If the repository list remains empty, ensure the specified Artifactory server URL and credentials (if provided) are valid.

Select the target repository you want Bamboo to deploy artifacts and build-info to.

Resolve artifacts from Artifactory

Check if you wish all dependency resolution to go through Artifactory.

Notice: this will override any external repository definition in Maven settings or POM files.

Resolution Artifactory Server URL

Select an Artifactory server.

Resolution repository

Resolver Username

Name of a user with read permissions on the target repository.

Resolver Password

Password of a user with read permissions on the target repository.

Artifactory Server URL

Select an Artifactory server.

Target Repository

Select a target deployment repository.

Deployer Username

Name of a user with deployment permissions on the target repository.

Deployer Password

The password of the user entered above.

Running Licence Checks

If you have an Artifactory Pro license, you can set the **Run License Checks** checkbox so that Artifactory will scan all dependencies used by the build to check for any license violations.

This feature offers the following options:

Send License Violation Notifications to	A list of email addresses of users who should receive license violation notifications.
--	--

Limit Checks to the Following Scopes	The Maven dependency scopes on which the license check should be run. If left empty, all scopes will be checked.
Include Published Artifacts	Indicates that any build artifacts that are dependencies for other build artifacts, should also be scanned for license violations
Disable Automatic License Discovery	Tells Artifactory not to try and automatically analyze and tag the build's dependencies with license information upon deployment. You can still attach license information manually by running Auto-Find from the build's Licenses tab in Artifactory.

Run License Checks (Requires Pro)

Check if you wish that automatic license scanning will occur after build is complete.

Send License Violation Notifications to

Whitespace-separated list of recipient addresses.

Limit Checks To The Following Scopes

Space-separated list of scopes.

Include Published Artifacts

Include the builds published module artifacts in the license violation checks if they are

Disable Automatic License Discovery

Tells Artifactory to not try and automatically analyze and tag the builds dependencies w
Auto-Find from the builds Licenses tab in Artifactory.

Black Duck Code Center Integration

If you have an Artifactory Pro license, and a Black Duck Code Center account you can use the [Artifactory Black Duck Code Center integration](#) to automatically initiate an open source component approval process, and proactively monitor for security vulnerabilities.

This feature offers the following options:

Code Center application name	The name of the Black Duck Code Center application that should be invoked.
Code Center application version	The Black Duck Code Center application version.
Send compliance report email to	A list of email addresses of users who should receive license violation notifications.
Limit checks to the following scopes	The Maven dependency scopes on which the compliance check should be run. If left empty, all scopes will be checked.

Include Published Artifacts	Indicates that any build artifacts that are dependencies for other build artifacts, should also be scanned for license violations
Auto create missing component requests	Automatically create missing components in Black Duck Code Center application after the build is completed and deployed in Artifactory.
Auto discard stale component requests	Automatically discard stale components in Black Duck Code Center application after the build is completed and deployed in Artifactory.

Run Black Duck Code Center compliance checks (requires Artifactory Pro)

Check if you wish that automatic Black Duck Code Center compliance checks will occur after the build is completed.

Code Center application name

Salamtak API

The existing Black Duck Code Center application name.

Code Center application version

1.0

The existing Black Duck Code Center application version.

Send compliance report email to:

admin@company.com

Input recipients that will receive a report email after the automatic Black Duck Code Center compliance checks finished.

Limit checks to the following scopes:

Compile

A list of dependency scopes/configurations to run Black Duck Code Center compliance checks on. If left empty all dependencies from all scopes will be checked.

Include published artifacts

Include the builds published module artifacts in the Black Duck Code Center compliance checks if they are also used as dependencies for other modules in this build.

Auto-create missing component requests

Auto create missing components in Black Duck Code Center application after the build is completed and deployed in Artifactory.

Auto-discard stale component requests

Auto discard stale components in Black Duck Code Center application after the build is completed and deployed in Artifactory.

The Artifactory Generic Resolve Task

The Generic Resolve task can be used in any job with any combination of tasks.

It lets you specify dependency patterns that should be downloaded from Artifactory through the creation of File Specs. Read more about File Specs [here](#).

Before version 2.2.0, specifying dependency patterns was possible through Legacy Patterns, which became deprecated in version 2.2.0

The Artifactory Generic Deploy Task

The Generic Deploy task can be used in any job with any combination of tasks, and is provided to offer minimal Build Info support for all types.

This task collects all available information regarding the build from Bamboo, and provides a deployment mechanism for the artifacts produced.

Adding the Generic Deploy task automatically deploys Build Info collected from the Published Artifacts declaration in addition to the artifacts themselves. Specifying artifact patterns to be deployed to Artifactory is done through the creation of File Specs. Read more about File Specs [here](#).

Before version 2.2.0, specifying artifact patterns was possible through Legacy Patterns, which became deprecated in version 2.2.0

Make sure to add the Generic Deploy task as a final step!

Using File Specs

File Spec are specified in JSON format. They are used in the **Generic Resolve** and **Generic Deploy** tasks, to specify the dependencies to be resolved from Artifactory or artifacts to be deployed to it.

You can use File Specs in one of the following ways:

1. Manage them in your SCM, and then during the build, have them pulled to the workspace with the other sources. If you choose this option, you should select the "File" option in the "Upload spec source" or "Download spec source" field and specify the relative path to the File Spec in your workspace.
2. Save the File Spec JSON as part of the job configuration. If you choose this option, you should select the "Job configuration" option in the "Upload spec source" or "Download spec source" field and specify the File Spec JSON content in your workspace in the "File path" field.

You can read the File Spec schema [here](#).

The Artifactory Deployment Task

The Bamboo Artifactory Plugin also supports Bamboo Deployment projects (read more about Deployment projects [here](#)).

The Artifactory Deployment task collects the build artifacts which are shared by the build plan associated with the deployment task, and uploads them to Artifactory.

The screenshot shows the configuration for the 'Artifactory Deployment' task. On the left, a task list includes 'Clean working directory task', 'Artifact download', and 'Artifactory Deployment' (highlighted). Below the list is an 'Add task' button. The main configuration area is titled 'Artifactory Deployment configuration' and includes a 'Task description' field with the value 'artifactory deployment'. There is a 'Disable this task' checkbox which is unchecked. The 'Artifactory Deployment' section contains the following fields: 'Artifactory Server URL' (http://localhost:8081/artifactory), 'Target Repository' (ext-release-local), 'Deployer Username' (admin), and 'Deployer Password' (masked with dots). Below the password field is a note: 'The password of the user entered above.' At the bottom of the configuration area are 'Save' and 'Cancel' buttons.

The "Artifacts Download" Task

The **Artifacts Download** task must be prior to the **Artifactory Deployment** task in the Deployment job flow.

The Artifacts Directory

We recommend configuring a subdirectory for the **Artifacts Download** task.

Running a Build

Once you have completed setting up a project builder you can run it. The Artifactory plug-in commences at the end of the build and:

1. Deploys all artifacts to the selected target repository in one go (as opposed to the deploy at the end of each module build, used by Maven/Ivy).
2. Deploys the Artifactory build-info to the selected server, which provides full traceability of the build in Artifactory, with links back to the build in Bamboo.

```
08-Aug-2010 12:30:07 [INFO] Attaching shaded artifact.
08-Aug-2010 12:30:07 [INFO]
08-Aug-2010 12:30:07 [INFO] --- maven-install-plugin:2.3:install (default-install) @ build-info-extractor-ivy ---
08-Aug-2010 12:30:07 [INFO] Installing /home/noan/Work/atlassian/bamboo/bamboo-artifactory-plugin/trunk/target/bamboo/home/xml-data/build-dir/MCBOB-DEF/build-info-extractor-ivy/target/build-info-extractor-ivy-1.0-SNAPSHOT.jar
/home/noan/.m2/repository/org/jfrog/buildinfo/build-info-extractor-ivy/1.0-SNAPSHOT/build-info-extractor-ivy-1.0-SNAPSHOT-sources.jar
08-Aug-2010 12:30:07 [INFO] Installing /home/noan/Work/atlassian/bamboo/bamboo-artifactory-plugin/trunk/target/bamboo/home/xml-data/build-dir/MCBOB-DEF/build-info-extractor-ivy/target/build-info-extractor-uber.jar
/home/noan/.m2/repository/org/jfrog/buildinfo/build-info-extractor-ivy/1.0-SNAPSHOT/build-info-extractor-uber.jar
08-Aug-2010 12:30:07 [INFO] Artifactory Build Info Recorder: Deploying artifacts to http://192.168.1.9:8080/artifactory
08-Aug-2010 12:30:07 [INFO] Deploying artifact: http://192.168.1.9:8080/artifactory/lib-snapshots-local/org/jfrog/buildinfo/build-info-client/1.4.x-SNAPSHOT/build-info-client-1.4.x-SNAPSHOT.pom
08-Aug-2010 12:30:08 [INFO] Deploying artifact: http://192.168.1.9:8080/artifactory/lib-snapshots-local/org/jfrog/buildinfo/build-info-api/1.4.x-SNAPSHOT/build-info-api-1.4.x-SNAPSHOT-sources.jar
08-Aug-2010 12:30:08 [INFO] Deploying artifact: http://192.168.1.9:8080/artifactory/lib-snapshots-local/org/jfrog/buildinfo/build-info-extractor-ivy/1.0-SNAPSHOT/build-info-extractor-ivy-1.0-SNAPSHOT-uber.jar
08-Aug-2010 12:30:16 [INFO] Deploying artifact: http://192.168.1.9:8080/artifactory/lib-snapshots-local/org/jfrog/buildinfo/build-info-extractor-maven3/1.0.x-SNAPSHOT/build-info-extractor-maven3-1.0.x-SNAPSHOT.pom
08-Aug-2010 12:30:16 [INFO] Deploying artifact: http://192.168.1.9:8080/artifactory/lib-snapshots-local/org/jfrog/buildinfo/build-info-extractor/1.4.x-SNAPSHOT/build-info-extractor-1.4.x-SNAPSHOT.pom
08-Aug-2010 12:30:17 [INFO] Deploying artifact: http://192.168.1.9:8080/artifactory/lib-snapshots-local/org/jfrog/buildinfo/build-info-api/1.4.x-SNAPSHOT/build-info-api-1.4.x-SNAPSHOT.pom
08-Aug-2010 12:30:17 [INFO] Deploying artifact: http://192.168.1.9:8080/artifactory/lib-snapshots-local/org/jfrog/buildinfo/build-info-extractor/1.4.x-SNAPSHOT/build-info-extractor-1.4.x-SNAPSHOT-sources.jar
08-Aug-2010 12:30:18 [INFO] Deploying artifact: http://192.168.1.9:8080/artifactory/lib-snapshots-local/org/jfrog/buildinfo/build-info-extractor-maven3/1.0.x-SNAPSHOT/build-info-extractor-maven3-1.0.x-SNAPSHOT-sources.jar
08-Aug-2010 12:30:18 [INFO] Deploying artifact: http://192.168.1.9:8080/artifactory/lib-snapshots-local/org/jfrog/buildinfo/build-info-extractor-ivy/1.0-SNAPSHOT/build-info-extractor-ivy-1.0-SNAPSHOT-sources.jar
08-Aug-2010 12:30:21 [INFO] Deploying artifact: http://192.168.1.9:8080/artifactory/lib-snapshots-local/org/jfrog/buildinfo/build-info-extractor-maven3/1.0.x-SNAPSHOT/build-info-extractor-maven3-1.0.x-SNAPSHOT.jar
08-Aug-2010 12:30:22 [INFO] Deploying artifact: http://192.168.1.9:8080/artifactory/lib-snapshots-local/org/jfrog/buildinfo/build-info-client/1.4.x-SNAPSHOT/build-info-client-1.4.x-SNAPSHOT.jar
08-Aug-2010 12:30:22 [INFO] Deploying artifact: http://192.168.1.9:8080/artifactory/lib-snapshots-local/org/jfrog/buildinfo/build-info-extractor/1.4.x-SNAPSHOT/build-info-extractor-1.4.x-SNAPSHOT.jar
08-Aug-2010 12:30:22 [INFO] Deploying artifact: http://192.168.1.9:8080/artifactory/lib-snapshots-local/org/jfrog/buildinfo/build-info-extractor/1.4.x-SNAPSHOT/build-info-extractor-1.4.x-SNAPSHOT.pom
08-Aug-2010 12:30:23 [INFO] Deploying artifact: http://192.168.1.9:8080/artifactory/lib-snapshots-local/org/jfrog/buildinfo/build-info-parent/1.4.x-SNAPSHOT/build-info-parent-1.4.x-SNAPSHOT.pom
08-Aug-2010 12:30:23 [INFO] Deploying artifact: http://192.168.1.9:8080/artifactory/lib-snapshots-local/org/jfrog/buildinfo/build-info-extractor-ivy/1.0-SNAPSHOT/build-info-extractor-ivy-1.0-SNAPSHOT.jar
08-Aug-2010 12:30:23 [INFO] Deploying artifact: http://192.168.1.9:8080/artifactory/lib-snapshots-local/org/jfrog/buildinfo/build-info-api/1.4.x-SNAPSHOT/build-info-api-1.4.x-SNAPSHOT.jar
08-Aug-2010 12:30:23 [INFO] Artifactory Build Info Recorder: Deploying build info ...
08-Aug-2010 12:30:23 [INFO] Deploying build info to: http://192.168.1.9:8080/artifactory/api/build
08-Aug-2010 12:30:23 [INFO]
08-Aug-2010 12:30:26 [INFO] Reactor Summary:
08-Aug-2010 12:30:26 [INFO]
08-Aug-2010 12:30:26 [INFO] JFrog Build-Info ..... SUCCESS [1.503s]
08-Aug-2010 12:30:26 [INFO] JFrog Build-Info Client ..... SUCCESS [2.485s]
08-Aug-2010 12:30:26 [INFO] JFrog Build-Info Extractor ..... SUCCESS [2.199s]
08-Aug-2010 12:30:26 [INFO] JFrog Build-Info Maven 3 Extractor ..... SUCCESS [4.969s]
08-Aug-2010 12:30:26 [INFO] JFrog Build-Info Ivy Extractor ..... SUCCESS [3.432s]
08-Aug-2010 12:30:26 [INFO]
08-Aug-2010 12:30:26 [INFO] BUILD SUCCESS
```

You can also link directly to the information in Artifactory from a build run view in Bamboo:

The screenshot shows a Bamboo build run view for job #53. The job status is 'Job: Default Job was successful'. Below the job summary, there are tabs for 'Job Summary', 'Tests', 'Artifactory Build Info', 'Commits', 'Artifacts', 'Logs', 'Metadata', and 'Artifactory Release & Promotion'. The 'Artifactory Build Info' tab is selected, and it displays the Artifactory logo and the text 'Artifactory Build Info'.

License

The Bamboo Artifactory plug-in is available under the Apache v2 License.

Attaching Searchable Parameters

You can define parameters that should be attached to build info and artifacts that are deployed by the plugin.

To define a parameter, under **Administration** go to **Build Resources | Global Variables**, fill in a **Key/Value** pair and click **Save**.

The available parameter types are:

- `buildInfo.property.*` - All properties starting with this prefix will be added to the root properties of the build-info.
- `artifactory.deploy.*` - All properties starting with this prefix will be attached to any produced artifacts that are deployed.

Using a Properties File

Instead of defining properties manually, you can point the plug-in to a properties file.

To do so, define a property named `buildInfoConfig.propertiesFile` and set its value to the absolute path of the properties file.

Global Variables

You can use this page to view, add, edit and delete global variables. Global variables are available on every build run in Bamboo and can be accessed using `$(bamboo.globalVarName)`

Key	Value	
<input type="text" value="artifactory.deploy.moo"/>	<input type="text" value="mcmod"/>	<input type="button" value="Add"/>
buildInfo.property.bob	mcbob	

The given path and file should be present on the machine that is running the build agent, not the server.

Overriding Plan values using Bamboo Variables

The Artifactory Plugin supports overriding various in the plan configuration like Deployer credentials, Resolver credentials, repositories etc.

If you wish to override any of the values specified in the table below, you need to configure them as Bamboo variables either through the UI or append them to the REST URL request as a query parameters.

When assigning any value to these Bamboo variables, it will override the job configuration.

Example with REST

```
curl -ubamboo-user:bamboo-password -XPOST  
"http://<BAMBOO  
HOST>:8085/rest/api/latest/queue/MVN-JOB?stage&executeAllStages&bamboo.var  
iable.artifactory.override.deployer.username=new_username&bamboo.variable.  
artifactory.override.deployer.password=new_password"
```

In the example above, we use CURL to remotely invoke a Bamboo plan. We set the Deployer username and Deployer password for this specific request.

Note that we add the "bamboo.variable" prefix to the query parameters.

Note that the sent values will be applied only if the specific task support them. For example: currently Artifactory Gradle tasks do not support Resolver credentials, hence those values will be ignored if sent.

Parameter name	Description	Supported jobs
artifactory.override.deployer.username	Deployer username	Maven, Gradle, Ivy, Generic deploy
artifactory.override.deployer.password	Deployer password	Maven, Gradle, Ivy, Generic deploy
artifactory.override.resolver.username	Resolver username	Maven, Generic resolve
artifactory.override.resolver.password	Resolver password	Maven, Generic resolve
artifactory.override.resolve.repo	Resolve repository	Maven, Gradle
artifactory.override.deploy.repo	Deploy repository	Maven, Gradle, Ivy, Generic deploy

artifactory.task.override.jdk	<p>If set to true, check the value of artifactory.task.override.jdk.env.var.</p> <p>If that variable is populated with an environment variable, use the value of that environment variable as the Build JDK path.</p> <p>If artifactory.task.override.jdk.env.var is not defined, use the value of JAVA_HOME for the Build JDK.</p>	Maven, Gradle, Ivy
artifactory.task.override.jdk.env.var	Stores the name of another environment variable whose value should be used for the build JDK.	Maven, Gradle, Ivy

Release Management

The Artifactory Plugin provides a powerful feature for release management and promotion. For details please refer to [Bamboo Artifactory Plugin - Release Management](#).

Push to Bintray

Bintray users can publish build artifacts to Bintray by using the Artifactory Bintray integration.

This can be done on the **Push to Bintray** tab of the Bamboo Artifactory Plugin in one of two ways:

1. You can configure your Bintray details in a [descriptor file](#) which should be added to your list of build artifacts
2. You can check the **Override descriptor file** checkbox and specify the details in the [Push to Bintray tab UI](#).

Using a Descriptor File

1. Create a descriptor file named *bintray-info.json*.

▼ [bintray-info.json](#). [Click for details...](#)

Here is an example:

```

{
- "repo": {
+ "name": "test",
- "type": "generic",
- "private": false,
- "premium": false,
- "desc": "My test repo",
- "labels": ["label1", "label2"],
- "updateExisting": false
},
+ "package": {
+ "name": "auto-upload",
+ "repo": "test",
+ "subject": "myBintrayUser",
- "desc": "I was pushed completely automatically",
- "website_url": "www.jfrog.com",
- "issue_tracker_url":
"https://github.com/bintray/bintray-client-java/issues",
+ "vcs_url": "https://github.com/bintray/bintray-client-java.git",
+ "licenses": ["MIT"],
- "labels": ["cool", "awesome", "gorilla"],
- "public_download_numbers": false,
- "public_stats": false,
- "attributes": [{"name": "att1", "values" : ["vall"], "type":
"string"},
                {"name": "att2", "values" : [1, 2.2, 4], "type": "number"},
                {"name": "att5", "values" : ["2014-12-28T19:43:37+0100"],
"type": "date"}]
},
+ "version": {
+ "name": "0.5",
- "desc": "This is a version",
- "released": "2015-01-04",
- "vcs_tag": "0.5",
- "attributes": [{"name": "VerAtt1", "values" : ["VerVall"], "type":
"string"},
                {"name": "VerAtt2", "values" : [1, 3.3, 5], "type": "number"},
                {"name": "VerAtt3", "values" : ["2015-01-01T19:43:37+0100"],
"type": "date"}],
- "gpgSign": false
},
"applyToFiles": ["repo1/org/jfrog/*.*",
"repo2/org/jfrog/test/module*/*.jar", "repo3/org/jfrog/test/**/*.*",
"repo2/org/jfrog/test/**/art.?ar"],
"applyToRepoFiles": ["/org/jfrog/*.*, jfrog/test/**/*.*"],
"applyToProps": [{"upload.prop1": ["vall", "val2"]}, {"upload.prop2":
["*"]}, {"*": ["valueRegardlessOfProperty"]}],
"publish": true
}

```

The file's name itself must contain the string **bintray-info** (anywhere in the name) and have a `.json` extension

Most of the fields are self-explanatory, however below are descriptions for those fields whose purpose may be less obvious:

Field	Purpose												
updateExisting	Signifies Artifactory should update an existing repository with the same name with the values in the jsondescriptor (applies only to the 'labels' and 'desc' fields)												
subject	Can either be your Bintray user name or the organization you are pushing to. The credentials that are used in the operation are those you defined in your user profile (or in the <i>default</i> section).												
applyToFiles	<p>If you are pushing a complete build, this field should remain empty.</p> <p>When pushing files, this field should contain a comma-separated list of files (in JSON format) that should be pushed. A file matching any of the file specifications will be pushed (i.e. an "OR" relationship).</p> <p>You may use wildcards as follows:</p> <ul style="list-style-type: none"> *: match any 0 or more characters ** : recursively match any sub-folders (in the path section only) ?: match any 0 or 1 character <p>Here are some examples of valid search paths:</p> <table border="1"> <thead> <tr> <th>Path</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td><code>repo1/org/jfrog/myTest.jar</code></td> <td>The file <code>myTest.jar</code> under <code>repo1/org/jfrog</code></td> </tr> <tr> <td><code>repo1/org/jfrog/*.*</code></td> <td>All files under <code>repo1/org/jfrog</code></td> </tr> <tr> <td><code>repo2/org/jfrog/test/module*/*.jar</code></td> <td>All <code>.jar</code> files under any subfolder of <code>repo2/org/jfrog/test</code> whose name starts with "module"</td> </tr> <tr> <td><code>repo2/org/jfrog/test/**/*.jar</code></td> <td>All <code>.jar</code> files under any subfolder of <code>repo2/org/jfrog/test</code></td> </tr> <tr> <td><code>repo2/org/jfrog/test/**/art.?ar</code></td> <td>All files named "art" with a file extension that has 2 or 3 characters and ends with "ar" under any subfolder of <code>repo2/org/jfrog/test</code></td> </tr> </tbody> </table>	Path	Meaning	<code>repo1/org/jfrog/myTest.jar</code>	The file <code>myTest.jar</code> under <code>repo1/org/jfrog</code>	<code>repo1/org/jfrog/*.*</code>	All files under <code>repo1/org/jfrog</code>	<code>repo2/org/jfrog/test/module*/*.jar</code>	All <code>.jar</code> files under any subfolder of <code>repo2/org/jfrog/test</code> whose name starts with "module"	<code>repo2/org/jfrog/test/**/*.jar</code>	All <code>.jar</code> files under any subfolder of <code>repo2/org/jfrog/test</code>	<code>repo2/org/jfrog/test/**/art.?ar</code>	All files named "art" with a file extension that has 2 or 3 characters and ends with "ar" under any subfolder of <code>repo2/org/jfrog/test</code>
Path	Meaning												
<code>repo1/org/jfrog/myTest.jar</code>	The file <code>myTest.jar</code> under <code>repo1/org/jfrog</code>												
<code>repo1/org/jfrog/*.*</code>	All files under <code>repo1/org/jfrog</code>												
<code>repo2/org/jfrog/test/module*/*.jar</code>	All <code>.jar</code> files under any subfolder of <code>repo2/org/jfrog/test</code> whose name starts with "module"												
<code>repo2/org/jfrog/test/**/*.jar</code>	All <code>.jar</code> files under any subfolder of <code>repo2/org/jfrog/test</code>												
<code>repo2/org/jfrog/test/**/art.?ar</code>	All files named "art" with a file extension that has 2 or 3 characters and ends with "ar" under any subfolder of <code>repo2/org/jfrog/test</code>												
applyToRepoFiles	<p>If you are pushing a complete build, this field should remain empty.</p> <p>When pushing files, this field should contain a comma-separated list of files (in JSON format) that should be pushed. A file matching any of the file specifications will be pushed (i.e. an "OR" relationship).</p> <p>This field behaves similarly to <code>applyToFiles</code>, including wildcards as described above, only it refers to relative paths inside the repo that contains the json descriptor file:</p> <ul style="list-style-type: none"> If the path starts with a leading '/' then the parent for this path is the repository's root, so the path <code>/org/jfrog/*.*</code> will actually point to <code>containingRepo/org/jfrog/*.*</code> If the path doesn't start with a '/' then the parent for this path is the folder containing the descriptor. So if the descriptor resides in <code>/org/jfrog/bintray-info.json</code>, the path <code>/test/myPackage/*.*</code> will actually point to <code>containingRepo/org/jfrog/test/myPackage/*.*</code> 												
applyToProps	<p>0 or more <code>key:value</code> pairs with which to filter the selected files by properties. The '*' and '?' wildcards are supported in this filter as well.</p> <p>A file matching all of the property specifications will be pushed (i.e. an "AND" relationship)</p>												
publish	If set to true, the version will be automatically published once the push operation is complete.												
gpgSign	<p>If set to true and no passphrase was passed as a parameter to the REST API call, Artifactory will attempt to sign the version without any passphrase.</p> <p>If you provide the <code>gpgPassphrase</code> parameter in the REST API call, this will cause the call to ignore this flag and the version will be signed with the passphrase that was passed.</p>												

2. Commit the descriptor file to your source control along with your project sources.
3. Modify your build script to attach the file to your build artifacts.

Using the "Push to Bintray" Tab UI

1. Check the **Override descriptor file** checkbox in the **Push to Bintray** tab.
2. Fill in the fields that are displayed.

Bintray Required Fields

For Bintray OSS users, all fields are mandatory.

For Bintray Pro accounts, the **Licenses** and **VCS URL** fields are optional .

Job Summary Artifactory Build Info Tests Commits Artifacts Logs Metadata Push to Bintray



Push to Bintray

Override descriptor file

Check this if you want to use Bintray override parameters.

Sign method

Decide if and how to sign your files.

GPG Passphrase

Maven Central Sync

Push to Bintray



Push to Bintray

Override descriptor file

Check this if you want to use Bintray override parameters.

Override descriptor

Subject

Subject in Bintray

Repository

A target repository in Bintray

Package name

A target package name under the repository.

You must first create the package in Bintray if it does not exist.

Version

A target version under the package. If the version does not yet exist in Bintray, it is created automatically

Licenses

Comma separated list of valid licenses.

For example: BSD, MIT, Apache-2.0

VCS URL

Please enter a valid Version Control URL.

This is not mandatory if you are a Premium Bintray user.

Sign method

Decide if and how to sign your files.

GPG Passphrase

Maven Central Sync

Push to Bintray

Maven Central sync with Bintray

When checking the "Maven Central Sync" checkbox in Push to Bintray configuration page your build will be published to Maven Central after it is pushed to Bintray.

Only packages included to [jcenter](#) can be synced with Maven Central automatically.

configuration

In order to use Maven Central sync you need to configure your Bintray and Sonatype OSS credentials in Artifactory plugin page like shown in the image below.

Bintray Configuration

The Bintray Configuration details are used for the "Maven Central Sync" option included as part of the "Push to Bintray" functionality

Bintray Username	<input type="text" value="bintray_user"/>
	<small>User name on Bintray</small>
Bintray Api Key	<input type="password" value="*****"/>
	<small>Users Api key for Bintray</small>
Sonatype OSS Username	<input type="text" value="maven_central_user"/>
	<small>Sonatype username for syncing with Maven Central directly from Bintray</small>
Sonatype OSS Password	<input type="password" value="*****"/>
	<small>Sonatype OSS Password for syncing with Maven Central directly from Bintray</small>
	<input type="button" value="Save"/> <input type="button" value="Test Bintray"/>

Bintray credentials

"Push to Bintray" works with Bintray credentials configured in Artifactory. You only need to specify Bintray credentials if you are using the Maven Central sync option.

Changelog

▼ [Click to see change log details](#)

2.3.0 (10 Oct 2017)

1. Support pattern exclusion in File Specs (BAP-391)
2. File specs AQL optimizations (BAP-395)
3. Dependencies repositories have been added to the plugin's maven descriptor (BAP-397)
4. Bug fixes (BAP-385, BAP-390, BAP-396)

2.2.0 (6 Aug 2017)

1. File Specs support for the Generic Resolve and Generic Deploy Tasks (BAP-377)
2. Upgrade JGit (BAP-381)
3. Bug fixes (BAP-378, BAP-379, BAP-380, BAP-382, BAP-383, BAP-384, BAP-387)

2.1.1 (22 Jun 2017)

1. Compatibility with Bamboo 6.0.x (BAP-376)

2.1.0 (20 Apr 2017)

1. Artifactory Release Management API changes (BAP-374)
2. Bug fixes (BAP-372, BAP-373)

2.0.2 (16 Feb 2017)

1. Compatibility with Bamboo 5.15.x (BAP-370)

2.0.1 (29 Jan 2017)

1. Bug fix (BAP-369)

1.10.2 (22 Sep 2016)

1. Bug fixes (BAP-360, BAP-359)

1.10.1 (7 Apr 2016)

1. Bug fixes (BAP-345)

1.10.0 (25 Feb 2016)

1. Compatibility with Bamboo 5.10.x (BAP-336)
2. Coordinate the deployment order of artifacts according to module info in the Gradle task (BAP-294)
3. Bug fix (BAP-303)

1.9.2 (22 Dec 2015)

1. Bug fixes (BAP-330, BAP-331)

1.9.1 (13 Dec 2015)

1. Bug fixe (BAP-312)

1.9.0 (26 Nov 2015)

1. Support sending parameters when invoking Bamboo Artifactory tasks remotely. (BAP-281, BAP-232)
2. New "Push to Maven Central" (BAP-284)
3. Bug fixes (BAP-313, BAP-306, BAP-290, BAP-288)

1.8.2 (27 Oct 2015)

1. Bug fixes (BAP-289, BAP-292, BAP-302)

1.8.1 (4 Aug 2015)

1. Bug fix (BAP-282)

1.8.0 (15 Jun 2015)

1. Add push to Bintray support (BAP-257)
2. Make Artifactory Upload Task available for Deployment projects (BAP-264)
3. Ability not to promote the version on Gradle Release Staging (BAP-258)
4. Bug fixes (BAP-270, BAP-269, BAP-267, BAP-266, BAP-261, BAP-260, BAP-254, BAP-246)

1.7.7 (30 Mar 2015)

1. Support for Bamboo 5.8.x (BAP-249)

1.7.6 (14 Jan 2015)

1. Support for Bamboo 5.7.x (BAP-230)
2. Compatibility with Maven 3.2.5 (BAP-244)
3. Enable overriding the Build JDK value using Bamboo variables (BAP-240)
4. Bug fix (BAP-241)

1.7.5 (10 Nov 2014)

1. Support Atlassian Stash source control management (BAP-206)
2. Artifactory generic Resolve task (BAP-207)
3. Maven 3 tasks - Record Implicit Project Dependencies and Build-Time Dependencies (BAP-225)

1.7.4 (12 Aug 2014)

1. Support for Bamboo 5.6 (BAP-218)
2. Bug fix (BAP-219)

1.7.3 (29 Jul 2014)

1. Add support for Gradle 2.0 (GAP-153)
2. Bug fix (BAP-212)

1.7.2 (25 Jun 2014)

1. Bug fixes (BAP-196, BAP-208, BAP-166)

1.7.1 (26 MAY 2014)

1. A new check box that gives the ability to ignore artifacts that are not deployed according to include/exclude patterns. (BAP-180)

1.7.0 (06 Apr 2014)

1. Fix Support for Bamboo 5.4+
2. Supporting Git Shared Credentials in Release Management functionality (BAP-189)
3. Adding Version Control Url property to the Artifactory Build Info JSON. (BAP-200)
4. Bug fixes (BAP-197)

1.6.2 (24 Nov 2013)

1. Fix Support for Bamboo 5.2
2. Add Artifactory BlackDuck integration
3. Bug fixes (BAP-182 BAP-184 BAP-186 BAP-184)

1.6.1 (03 Oct 2013)

1. Support form Bamboo 5.1.1
2. Bug fixes 1.6.1

1.6.0 (16 Jul 2013)

1. Support form Bamboo 5.0

1.5.6 (03 Sep 2013)

1. Support form Bamboo 4.2

1.5.5 (03 Sep 2012)

1. Support for include/exclude captured environment variables (BAP-143)
2. Bug fixes (MAP-41 MAP-40 GAP-129 BAP-148 IAP-32)

1.5.4 (25 Jun 2012)

1. Support Bamboo 4.1.
2. Bug fixes. (JIRA)

1.5.3 (02 Apr 2012)

1. Support Bamboo 4.0.

1.5.2 (02 Apr 2012)

1. Support Perforce for release management. (BAP-133)
2. Bug fixes. (JIRA)

1.5.1 (05 Jan 2012)

1. Compatible release plugin for version 3.4.2. (BAP-116)
2. Support for Gradle properties deployment. (BAP-117)
3. Unique icon for each Artifactory task type.
4. Setting Bamboo job requirements correctly for all builder types. (BAP-125)

1.5.0 (11 Dec 2011)

1. Compatible with bamboo version 3.3.x.
2. Compatible with Gradle 1.0-milestone-6.

1.4.2 (19 Sep 2011)

1. Bug fix (BAP-91)

1.4.1 (01 Aug 2011)

1. Support for Bamboo 3.2.x
2. Bug fix (BAP-90)

1.4.0 (14 Jul 2011)

1. Introducing Release Management capabilities.
2. Generic Build Info support for all job types.
3. Bug fixes.

1.3.2 (14 Jun 2011)

1. Bug fix (BAP-65)

1.3.1 (13 Jun 2011)

1. Bug fix (BAP-64)

1.3.0 (30 May 2011)

1. Support for Bamboo 3.1.x

1.2.0 (2 Mar 2011)

1. Support for Bamboo 3.x

1.1.0 (2 Jan 2011)

1. Gradle Support - Gradle builds are now fully supported with the new Gradle builder
2. Ivy builds now support custom Ivy patterns for artifacts and descriptors
3. Support for Bamboo 2.7.x

1.0.3 (21 Nov 2010)

1. Add Include/exclude pattern for artifacts deployment
2. Bug fix (BAP-26)

1.0.2 (7 Nov 2010)

1. Control for including published artifacts when running license checks
2. Limiting license checks to scopes
3. Control for turning off license discovery when running license checks

Artifactory Generic Resolve

Artifactory Server URL

http://10.0.0.126:8080/artifactory ▼

Select an Artifactory server.

Resolution repository

libs-snapshot ▼

Resolver Username

admin

Name of a user with read permissions on the target repository.

Resolver Password

.....

Password of a user with read permissions on the target repository.

Edit Resolved Artifacts

```
libs-release-local:org/jfrog/test/multi/maven-metadata.xml  
libs-release-local:**/*sh@generic free style#LATEST
```

Save

Cancel

Artifactory Generic Deploy configuration

Task description

|

Disable this task

Artifactory Generic Deploy

Artifactory Server URL

▼

Select an Artifactory server.

Save

Cancel

Overview

Artifactory supports release management through the [Bamboo Artifactory Plugin](#).

When you run your builds using [Maven](#) or [Gradle](#) with jobs that use [Git](#) or [Perforce](#) as your version control system, you can manually stage a release build allowing you to:

- Change values for the release and next development version
- Choose a target staging repository for deployment of the release, and
- Create a VCS tag for the release.

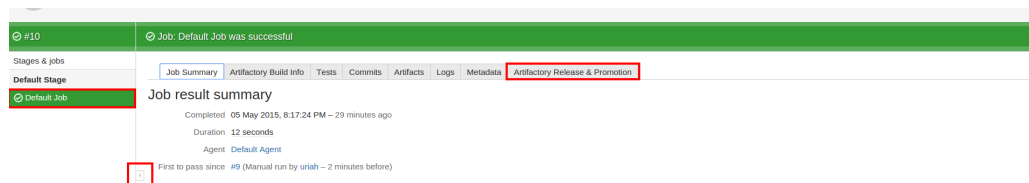
Staged release builds can later be **promoted** or **rolled-back**, changing their release status in Artifactory and, optionally, moving the build artifacts to a different target repository.

Inside Artifactory, the history of all build status change activities (staged, promoted, rolled-back, etc.) is **recorded and displayed** for full traceability.

When release management is enabled, the Artifactory release staging link appears on the top header bar in the job page.

Displaying the Release and Promotion Tab

To display the **Artifactory Release & Promotion** tab you need to click the small arrow indicated below.



Release management tab moved from plugin version 1.7.0

From Bamboo Artifactory Plugin version 1.7.0, the Release Management tab was moved from the **Plan** page level to the **Job** page level because the process applies to artifacts in the context of a single job rather than a whole plan (which may hold several jobs).

The tab name was also changed from **Artifactory Release management** to **Artifactory Release & Promotion**.

Page Contents

- [Overview](#)
- [Maven Release Management](#)
 - [Configuring Maven Jobs](#)
 - [Staging a Maven Release Build](#)
- [Gradle Release Management](#)
 - [Configuring Gradle Jobs](#)
 - [Staging a Gradle Release Build](#)
- [Promoting a Release Build](#)
- [Working with Git](#)
- [Working with Perforce](#)

Maven Release Management

The [Bamboo Artifactory Plugin](#) manages a release with Maven running the build only once using the following basic steps:

1. Change the POM versions to the release version (before the build starts).
2. Trigger the Maven build (with optionally different goals).
3. Commit/push changes to the release branch.
4. Change the POM versions to the next development version.
5. Commit/push changes to the trunk.

If the build fails, the plugin attempts to rollback the changes (both local and committed).

For more information including configuration of Maven Runners, and Jobs and staging a release build, please refer to [Bamboo Artifactory Plugin](#).

Configuring Maven Jobs

To enable release management in Maven jobs, edit the job configuration and check the **Enable Artifactory release management** checkbox.

Enable Release Management

Enable Release Management to Artifactory

VCS Tags Base URL/Name

This is the name of the tag/label.

Git Release Branch Name Prefix

The prefix of the release branch name (applicable only to Git).

Alternative Maven Tasks and Options

Alternative Maven and options to execute for a Maven build running as part of the release. If left empty, the build will use original tasks and options instead of replacing them.

VCS Type

Git URL*

VCS Authentication Type

Git Username*

Git Password*

Staging a Maven Release Build

Clicking on the release staging link opens a new page with configuration options for the release build:



Artifactory Pro Release Staging

- Module Version Configuration
- One version for all modules.
 - Version per module
 - Use existing module versions

Release Value

Next Integration Value

VCS Configuration

Create VCS Tag

Tag URL/name:

Tag comment

Next development version comment

Publishing Repository

Staging Comment:

The release staging page displays the last version built (the version tag is that of the root POM, and is taken from the last build that is not a release). Most of the fields in the form are populated with default values.

Version configuration controls how the plugin changes the version in the POM files (global version for all modules, version per module or no version changes).

If the **Create VCS tag** checkbox is checked (default), the plugin commits/pushes the POMs with the release version to the version control system with the commit comment. When using Git, there's also an option to create a release branch.

Click on the **Build and Release to Artifactory** button to trigger the release build.

Target server is Artifactory Pro?

If the target Artifactory server is a Pro edition, you can change the target repository, (the default is the release repository configured in Artifactory publisher) and add a staging comment which is included in the build info deployed to Artifactory.

The [Bamboo Artifactory Plugin](#) supports release management when running builds with Gradle. This relies on the version property (and others) managed by the `gradle.properties` file. The plugin reads the properties from the Artifactory release management configuration, and modifies those properties in the `gradle.properties` file.

The plugin manages a release using the following basic steps:

1. Modify properties in the `gradle.properties` to release values (before the build starts).
2. Trigger the Gradle build (with optionally different tasks and options).
3. Commit/push changes to the release branch.
4. Modify the `gradle.properties` to the next integration values.
5. Commit/push changes to the trunk.

Configuring Gradle Jobs

To enable Gradle release management, edit the Artifactory Gradle Task configuration and check the **Enable Release Management** checkbox.

Enable Release Management

Enable Release Management to Artifactory

VCS Tags Base URL/Name

This is the name of the tag/label.

Git Release Branch Name Prefix

The prefix of the release branch name (applicable only to Git).

Release Properties

Properties in your projects `gradle.properties` file whose value should change upon release.

Next Integration Properties

Properties in your projects `gradle.properties` file whose value should change upon release, but also for work on the next integration/development version after the release has been created.

Alternative Gradle Tasks and Options

Alternative tasks and options to execute for a Gradle build running as part of the release. If left empty, the build will use original tasks and options instead of replacing them.

VCS Type

Git URL*

VCS Authentication Type

Git Username*

Git Password*

Staging a Gradle Release Build

Once release management is enabled, the Artifactory **Release staging** tab appears in the top header bar on the job page.

Clicking on the **Release staging** tab opens a new page with configuration options for the release build:



Artifactory Pro Release Staging

Property Key	<input type="text" value="wharf-core-version"/>
Current Value	<input type="text" value="1.49-SNAPSHOT"/>
Release Value	<input type="text" value="1.49"/>
Next Integration Value:	<input type="text" value="1.50-SNAPSHOT"/>

VCS Configuration

	<input checked="" type="checkbox"/> Use Release Branch:
Release branch:	<input type="text" value="REL-BRANCH-1.49"/>
	<input checked="" type="checkbox"/> Create VCS Tag
Tag URL/name:	<input type="text" value="1.49"/>
Tag comment	<input type="text" value="[artifactory-release] Release version 1.49"/>
Next development version comment	<input type="text" value="[artifactory-release] Next development version"/>
Publishing Repository	<input type="text" value="libs-release-local"/>
	Select a publishing repository.
Staging Comment:	<input type="text"/>

The **Release staging** tab displays the **Release** and **Next development** properties configured for the job. The plugin reads these values from the `gradle.properties` file and attempts to calculate and display **Release** and **Next integration version** in the text fields.

If **Create VCS tag** is checked (default), the plugin commits/pushes the POMs with the release version to the version control system with the commit comment. When using Git, if **Use release branch** is checked, the **Next release version** changes are carried out on the release branch instead of the current checkout branch. The final section allows you to change the target repository (the default is the release repository configured in Artifactory publisher) and an optional staging comment which includes the build info deployed to Artifactory.

Click on the **Build and Release to Artifactory** button to trigger the release build.

Promoting a Release Build

You can promote a release build after it completes successfully.

This is not a mandatory step but is very useful because it allows you to mark the build as released in Artifactory, and move or copy the built artifacts to another repository so they are available to other users.

To promote a build, browse to the build's result page and click the **Artifactory Release & Promotion** tab.

Artifactory Pro required

Promotion features are only available with Artifactory Pro

Job Summary Tests Artifactory Build Info Commits Artifacts Logs Metadata **Artifactory Release & Promotion**



Artifactory Release Promotion

Promotion Mode Normal

Target Status

Comment

Target promotion repository

Select a promotion repository.

Include dependencies

Use Copy

Update

Select the target status of the build ("Released" or "Rolled-Back"). You may also enter a comment to display in the build in Artifactory.

To move or copy the build artifacts, select the **Target promotion repository**.

Release management

From Bamboo Artifactory Plug-in version 1.7.0, the Artifactory Release Promotion was moved from the **Artifactory** tab to the new **Artifactory Release & Promotion** tab.

Working with Git

To work with Git, the Git plugin must be configured to build one branch **AND** to checkout to the same local branch.

The remote URL should allow Read+Write access.

The [Bamboo Artifactory Plugin](#) uses the Git client installed on the machine and uses its credentials to push back to the remote Git repository.

Source Repository

Source Control

Repository URL * ?
The URL of Git repository.

Branch
The name of the branch (or tag) containing source code.

Authentication Type

SSH Key
SSH private key you want to use to access the repository.

SSH Passphrase

Passphrase you want to use to access SSH private key.

Use shallow clones
Fetches the shallowest commit history possible. Do not use if your build depends on full repository history.

Source Repository

Source Control

Username *
The GitHub user required to access the repositories.

Change password?

Repository
Select the repository you want to use for your Plan.

Branch
Choose a branch you want to check out your code from.

Use shallow clones
Fetches the shallowest commit history possible. Do not use if your build depends on full repository history.

During the release, the plugin performs the following steps:

1. If **Create Branch** is checked, create and switch to the release branch.
2. Commit the release version to the current branch.
3. Create a release tag.
4. Push the changes.
5. Switch to the checkout branch and commit the next development version.
6. Push the next development version to the working branch

Shallow Clones

Bamboo's Git plugin allows the use of shallow clones, however this causes the "push" not to work.

Therefore, when using the Artifactory Bamboo Plugin, you must have shallow clones **unchecked**.

For more information about shallow clones, please refer to [git-clone Manual Page](#).

Release management with [Bamboo Artifactory Plugin](#) supports Perforce when using one checkout directory.

During the release the plugin does the following:

1. Commits the release version directly to the tag (if **Create VCStag** is checked). The release version is not committed to the working branch.
2. Commits the next development version to the working branch

Changes

Changes are only committed if the files are modified (POM files or `gradle.properties`).

MSBuild Artifactory Plugin

Overview

Artifactory brings Continuous Integration to MSBuild, TFS and Visual Studio through the MSBuild Artifactory Plugin. This allows you to capture information about deployed artifacts, resolve Nuget dependencies and environment data associated with MSBuild build runs, and deploy artifacts to Artifactory. In addition, the exhaustive build information captured by Artifactory enables fully traceable builds.

MSBuild Artifactory Plugin

The MSBuild Artifactory plugin is an [open source project on GitHub](#) which you can freely browse and fork.

Sample code

To get yourself started, [here is an example](#) of a solution with multiple projects that use the MSBuild Artifactory Plugin

The MSBuild Artifactory Plugin can be used whether you are running standalone builds or using a CI server. In either case, you should note the following points:

1. Standalone Integration

The MSBuild Artifactory Plugin fully integrates with the MSBuild process, so it can run as part of a standard build.

The plugin uses a conventional MSBuild XML configuration file to influence different stages of the build process.

2. CI Server Integration

When running MSBuild builds in your continuous integration server, using the plugin is transparent since it is effectively an integral part of the MSBuild process. The only difference is that the plugin collects information from the CI server.

The MSBuild Artifactory Plugin fully supports TFS and collects exhaustive build information to enable fully traceable builds. Support for additional CI servers such as Jenkins, TeamCity and Bamboo is partial, and the build information collected when running with these tools is correspondingly partial.

Page Contents

- [Overview](#)
 - [MSBuild Artifactory Plugin](#)
- [Installation](#)
- [Update](#)
- [Uninstalling](#)
- [Migration from the old plugin implementation](#)
- [Configuration](#)
 - [General Information](#)
 - [Resolution](#)
 - [Deployment](#)
 - [Checksum Deployment](#)
 - [Project-specific](#)

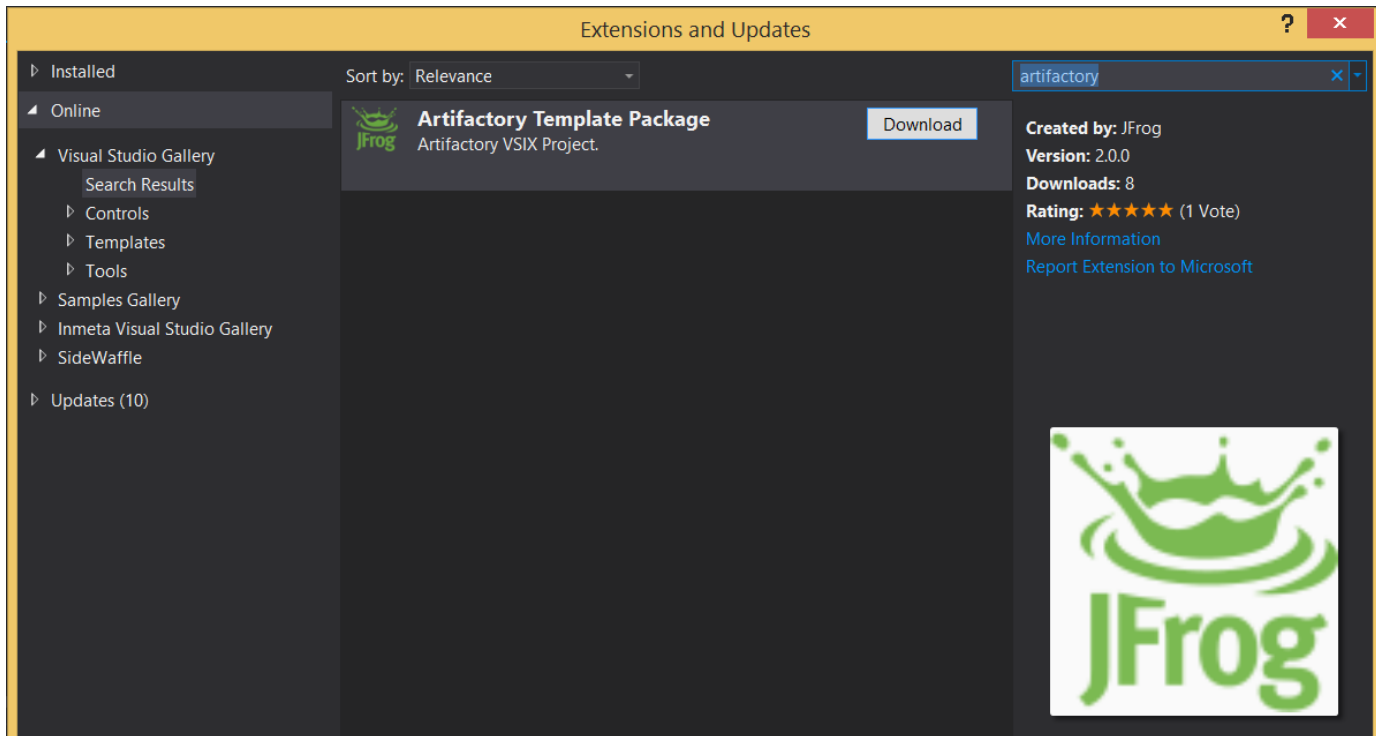
Deployment

- Environment Variables
- License Control
- Black Duck Code Center Integration
- Network Configuration
 - Deploying via Proxy
- Running a build with MSBuild Artifactory Plugin
- Team Foundation Server (TFS) Integration
 - MSBuild Arguments in TFS
 - Package Restore with Team Foundation Build
- License
- Screencast
- Changelog

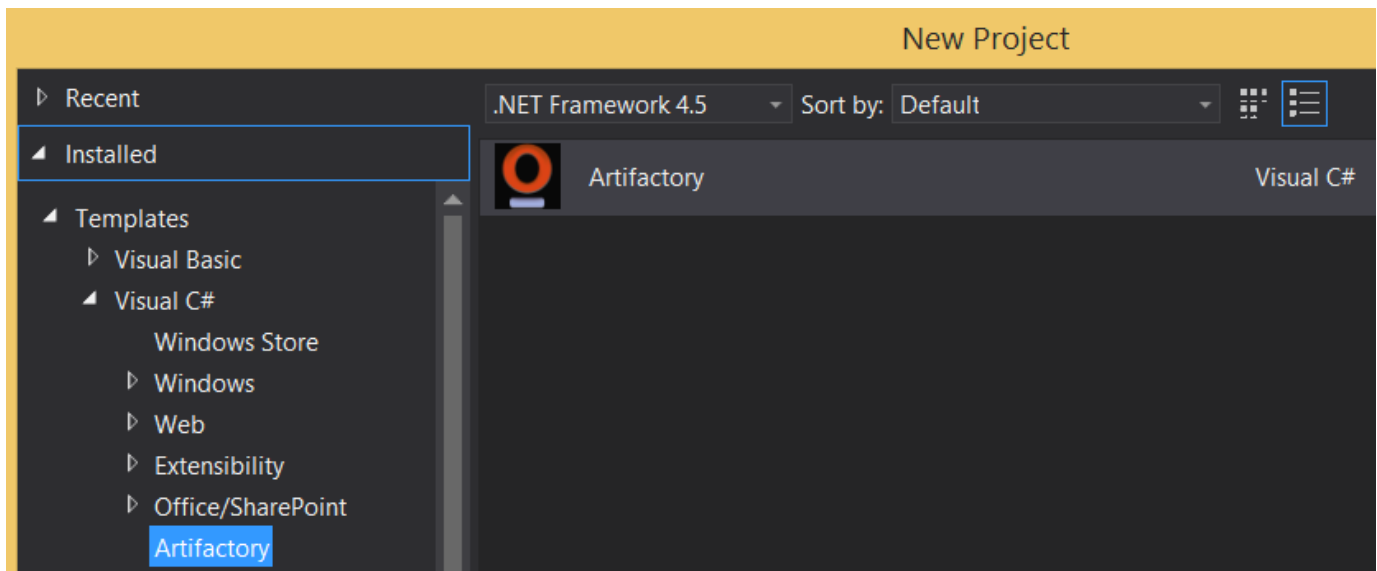
Installation

The MSBuild Artifactory Plugin is installed as a "Project Template" using Visual Studio as follows:

- Under **Tools**, choose **Extensions and Updates...**, select the **Visual Studio Gallery** source under the **Online** section, and run a search for "Artifactory".
- Select **Artifactory Template Package** extension found, and click **Install**.



- Once the installation is complete, select the **Solution** into which you want to install the plugin. right-click the solution node and select **Add | New Project....** Select **Artifactory** and click OK.

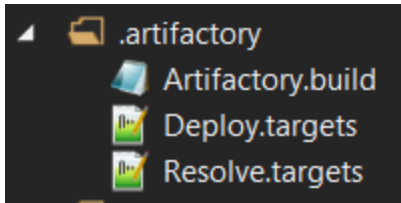


Creating Artifactory project

To resolve [Nuget](#) dependencies, the plugin requires a `.nuget` directory within your Solution. Before you start installation, make sure that this directory exists.

You should see the following changes to your Solution:

- A new custom project linked to the Artifactory plugin.
- A `.artifactory` folder is added to your Solution. Make sure this directory is committed to source control.
- Under the `.artifactory` folder, an `Artifactory.build` file is created. This is the main plugin configuration file.
- Two `.targets` files are created. These are used internally by the plugin and should not be modified manually.



- New [NuGet](#) packages related to the plugin should have been installed into the Solution.
- The plugin also imports its own MSBuild configuration file to the *artifactory.csproj* and *.nuget\NuGet.targets* files in order to be added to the MSBuild process.

The last step will be to link your relevant projects to the Artifactory project via the added **Project Reference**. Only the linked projects will be monitored by the plugin.

Installing plugin dependencies

The following two dependencies: [NuGet.core v2.8.2](#) and [Microsoft.Web.Xdt v2.1.1](#) must also be installed together with the plugin.

If you are installing the plugin from an instance of Artifactory, you need to ensure that Artifactory has access to the plugin and its dependencies. For example, you might have a virtual repository that references a local repository containing the plugin, and a remote repository that references nuget.org. For more details on configuring Artifactory, please refer to [NuGet Repositories](#).

Update

To update Artifactory extensions, execute the following steps:

- Under **Tools**, select **Extensions and Updates...**, and then, the "Visual Studio Gallery" source under the **Updates** section.
- Select the **Artifactory Template Package** extension found, and click **Update**.

Existing Project (Optional Step)

If you already have an existing Artifactory project template in your solution, and you want to update it to the latest one, execute the following steps:

- Right-click on the Artifactory project and select **Manage NuGet Packages...**
- In the **Manage NuGet Packages** window, select a source (e.g. nuget.org) under the **Updates** section.
- Select the **Artifactory** package found, and click **Update**.

Uninstalling

- Remove the *.artifactory* folder and its contents from the solution.
- Remove the custom Artifactory project from the solution.

Make sure to delete the items from the file system also.

Migration from the old plugin implementation

To migrate from the old plugin implementation, you need to uninstall it and then install the new implementation

- Follow the steps described in [Uninstalling](#).
- Follow the steps described in [Installation](#).

Configuration

General Information

The MSBuild Artifactory Plugin is configured in the *Artifactory.build* configuration file. The file is structured using **MSBuild** language conventions, so all the properties can be externally overridden using **Reserved Properties** or **Environment Properties**.

MSBuild can collect properties that were configured in the build scope, or in the Environment Variables. This ability can be helpful in different cases:

- You can dynamically override the plugin configuration according to the build context that it runs in.
- You can prevent sensitive information from being checked into source control.
For example, if the build runs under a build server such as TFS, all the Artifactory credentials can be defined by the server administrator, and will therefore, not be Checked in/Committed to source control.

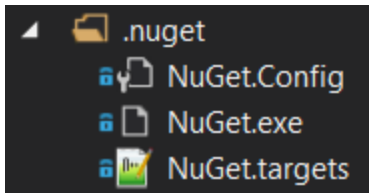
For more information about MSBuild properties, please refer to the [MSBuild Reference Documentation](#).

Configuration Instructions

For more details on configuration, please refer to the [Artifactory.build configuration file](#).

Resolution

To resolve packages, the MSBuild Artifactory Plugin uses the [NuGet Package Restore](#) feature with the [MSBuild-Integrated Package Restore](#) approach. To support this, the project that installed the plugin must be a part of a solution with the `.nuget` folder. If the `.nuget` folder is absent, the plugin will not override the Package Restore in the solution.



Please note the following points:

- Even though the plugin is installed on a project level, it overrides the NuGet resolution on **all** the projects under the same solution.
- Manual configuration in the `.nuget/NuGet.Config` file is ignored by the plugin.
- Modifying the `.nuget/NuGet.targets` file can cause unexpected behavior in the resolution process. We strongly recommend that you do not modify these files manually.
- The `.nuget` folder must be committed to source control.

Ensuring package resolution through Artifactory

In order to mitigate situations in which a network connection is not available, the NuGet client locally caches any artifacts downloaded from a remote repository in the **NuGet Local Cache** (under `%AppData%\Local\NuGet\Cache`). Subsequently, the NuGet client first checks the cache when trying to resolve packages. Therefore, artifacts downloaded from a remote repository in Artifactory or from the NuGet Gallery, typically get stored in this local cache and will be provided from the cache next time you try to reference them.

To ensure that the NuGet client resolves packages through Artifactory, you need to delete the NuGet Local Cache.

Deployment

In order to support a wide variety of project templates, solution structures and artifact types, the MSBuild Artifactory Plugin is designed to be very flexible and allows the user great freedom in configuring how to deploy packages.

- Using an **Input Pattern**, the user can specify the path to files that the plugin will collect for deployment. The path is relative to the project in which the plugin is installed, and to other projects referenced by it in the solution.
- Using an **Output Pattern**, the user can specify a deployment path in Artifactory that corresponds to the specified **Input Pattern**.
- The user can also specify **Custom Properties** that should annotate all the artifacts resulting from the specified **Input Pattern**.

Target repository layout

You may define a custom layout for your target repository, but it is up to you to specify the right Output Pattern to ensure that your artifacts are deployed to the right location within the repository. For more details, please refer to [Local Repositories](#).

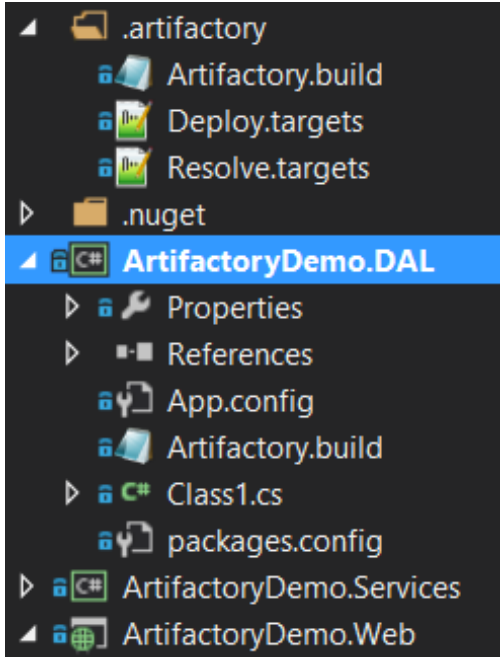
Checksum Deployment

To support Artifactory's "[Once-And-Only-Once](#)" [Content Storage](#), the plugin efficiently deploys packages to Artifactory using Checksum Deployment. Before an artifact is actually deployed, the plugin passes its checksum to Artifactory.

If the package already exists then Artifactory does not accept a new copy, it just creates a new metadata entry in the database to indicate that another "copy" of the artifact exists in specified deployment path.

Project-specific Deployment

The *Artifactory.build* file under the *.artifactory* directory applies to all projects within the solution. However, you can override the deployment configuration for a specific project providing an *Artifactory.build* file within the project scope. For example, you could use this to specify the full path of an artifact that needs to be deployed. The plugin detects the project-specific *Artifactory.build* file and applies the deployment configuration to that project, overriding the general deployment configuration. The example below shows the "ArtifactoryDemo.DAL" project with its own *Artifactory.build* file to override the general solution deployment configuration.



Environment Variables

You can enable the **EnvironmentVariables** tag so that MSBuild Artifactory Plugin uses all environment variables accessible by the build process and registers them in the build info. If running under a build server, the server's properties also used and registered. You may define **IncludePatterns** and **ExcludePatterns** to control which variables are included in the published build info.

Pattern wildcards

A pattern may contain the * and the ? wildcards. Include patterns are applied before exclude patterns.

Extensive build information may slow down deployment

Including all environment variables as part of the captured build information may result in very large build objects which in turn, may slow down deployment.

License Control

MSBuild Artifactory Plugin supports Artifactory Pro [License Control](#). This feature is controlled by several tags in the *Artifactory.build* configuration file.

LicenseControl	Enables or disables the license control feature
LicenseViolationRecipients	Specifies addresses of recipients that should receive license alerts by email

AutomaticLicenseDiscovery	When set, Artifactory will analyze and tag the build's dependencies with license information upon deployment <div style="border: 1px solid #f0e68c; padding: 10px;"> <p>Resource intensive Automatic license discovery is a resource intensive operation which may slow down deployment. If you do not run automatic license discovery, you can still attach license information manually by running 'Auto-Find'</p> <p>from the build's Licenses tab in the Artifactory UI. For more details, please refer to Examining Build Licenses.</p> </div>
IncludePublishedArtifacts	License checks are usually only run on the dependencies of the published package. If this flag is set, a license check is run on the deployed artifact itself (only valid for NuGet packages)
ScopesForLicenseAnalysis	Lets you specify the scopes on which license analysis should be run.

Black Duck Code Center Integration

If you are using Artifactory Pro and have an account with [Black Duck](#) Code Center, you can run the build through an automated, non-invasive, open source component approval process, and monitor for security vulnerabilities.

BlackDuckComplianceCheck	Enables or disables the license control feature
CodeCenterApplicationName	The existing Black Duck Code Center application name
CodeCenterApplicationVersion	The existing Black Duck Code Center application version
LicenseViolationRecipients	Specifies addresses of recipients that should receive license alerts by email
ScopesForLicenseAnalysis	Lets you specify the scopes on which license analysis should be run.
IncludePublishedArtifacts	License checks are usually only run on the dependencies of the published package. If this flag is set, a license check is run on the deployed artifact itself (only valid for NuGet packages)
AutoCreateMissingComponent	Auto create missing components in Black Duck Code Center application after the build is completed and deployed in Artifactory.
AutoDiscardStaleComponent	Auto discard stale components in Black Duck Code Center application after the build is completed and deployed in Artifactory.

Network Configuration

Deploying via Proxy

MSBuild Artifactory Plugin supports deployments via your network proxy. If the values in the **ProxySettings** tag of the *Artifactory.build* configuration file are not recognized by the plugin, it will fall back to using the **http_proxy** Environment Variables for proxy configuration using the format `http://<username>:<password>@proxy.com`.

You can bypass the proxy by setting the **Bypass** tag in the plugin configuration.

Resolution Proxy

Due to a [technical issue in the Nuget Client](#), you cannot configure the NuGet client for resolution via a proxy through the plugin. For the Nuget client to resolve artifacts via a proxy, you need to configure the proxy settings in `%APPDATA%\NuGet\NuGet.Config`.

For more information on how to configure a Nuget proxy, please refer to [NuGet Config Settings](#).

Running a build with MSBuild Artifactory Plugin

Once you have completed setting up the MSBuild Artifactory Plugin, you can run a project build. The plugin takes effect at the end of the build and does the following:

1. Publishes the specified published artifacts to the selected target repository and applies the proper path mappings.
2. Deploys the BuildInfo to Artifactory, providing [full traceability of the build](#), with links back to the build in TFS.

The example below shows Visual Studio output of a build log (minimum **verbosity** log level: Normal) with some deployed artifacts. At the bottom there is a link to the Build Info report on Artifactory.

```
Output
Show output from: Build
> Found packages.config. Using packages listed as dependencies
> Successfully created package 'bin\ \ArtifactoryDemo.Web.1.0.0.0.symbols.nupkg'.
>[Artifactory]:
> [Artifactory] Artifactory Post-Build task started
> [Artifactory] Processing build info...
> [Artifactory] Processing build modules...
> [Artifactory] Deploying artifact: http://192.168.56.1:8080/artifactory/nuget-local/nuget/ArtifactoryDemo.Web.1.0.0.0.nupkg
> [Artifactory] Deploying artifact: http://192.168.56.1:8080/artifactory/nuget-local/nuget/ArtifactoryDemo.Web.1.0.0.0.symbols.nupkg
> [Artifactory] Deploying artifact: http://192.168.56.1:8080/artifactory/nuget-local/general/ArtifactoryDemo.DAL.dll
> [Artifactory] Deploying artifact: http://192.168.56.1:8080/artifactory/nuget-local/general/ArtifactoryDemo.Web.dll
> [Artifactory] Deploying artifact: http://192.168.56.1:8080/artifactory/nuget-local/general/ArtifactoryDemo.Web.dll
> [Artifactory] Deploying artifact: http://192.168.56.1:8080/artifactory/nuget-local/general/AWSSDK.dll
> [Artifactory] Deploying artifact: http://192.168.56.1:8080/artifactory/nuget-local/general/Castle.Windsor.dll
> [Artifactory] Deploying artifact: http://192.168.56.1:8080/artifactory/nuget-local/general/EntityFramework.dll
> [Artifactory] Deploying artifact: http://192.168.56.1:8080/artifactory/nuget-local/general/EntityFramework.SqlServer.dll
> [Artifactory] Deploying artifact: http://192.168.56.1:8080/artifactory/nuget-local/general/Newtonsoft.Json.dll
> [Artifactory] Deploying artifact: http://192.168.56.1:8080/artifactory/nuget-local/general/System.Net.Http.Formatting.dll
> [Artifactory] Deploying artifact: http://192.168.56.1:8080/artifactory/nuget-local/general/System.Web.Http.dll
> [Artifactory] Deploying artifact: http://192.168.56.1:8080/artifactory/nuget-local/general/System.Web.Mvc.dll
> [Artifactory] Deploying artifact: http://192.168.56.1:8080/artifactory/nuget-local/general/System.Web.Optimization.dll
> [Artifactory] Deploying artifact: http://192.168.56.1:8080/artifactory/nuget-local/general/System.Web.Razor.dll
> [Artifactory] Deploying artifact: http://192.168.56.1:8080/artifactory/nuget-local/general/System.Web.WebPages.Razor.dll
> [Artifactory] Uploading build info to Artifactory...
> [Artifactory] Build successfully deployed. Browse it in Artifactory under http://192.168.56.1:8080/artifactory/webapp/builds/ArtifactoryDemo.Web/1411988146/2614-09-29T11:42:20.20+03:00/
>
>Build succeeded.
>
>Time Elapsed 00:00:07.86
***** Rebuild All: 3 succeeded, 0 failed, 0 skipped *****
```

Team Foundation Server (TFS) Integration

MSBuild Artifactory Plugin brings CI Build Integration to TFS users allowing you to efficiently deploy your artifacts to Artifactory. In addition to the BuildInfo that the plugin already registers, all parameters associated with TFS are also recorded to facilitate fully traceable builds.

For more information about how build information is used in Artifactory, please refer to [BuildInfo](#).

MSBuild Arguments in TFS

The MSBuild Artifactory Plugin configuration file supports MSBuild **Reserved Properties** or **Environment Properties**, and the best practice is to define these properties in the TFS build configuration. This lets you protect sensitive information and run the same build with different properties. Below is an example of properties configured in the TFS build definition.

Team Foundation Build uses a build process template defined by a Windows Workflow (XAML) file. The behavior of this template can be customized by setting the build process parameters provided by the selected template.

Build process template: **Default Template** Show details

Build process parameters:

2. Configurations	Any CPU Debug
3. Clean build	True
4. Output location	SingleFolder
5. Advanced	
MSBuild arguments	/p:ARTIFACTORY_USER="admin",ARTIFACTORY_PASSWORD="password"
MSBuild platform	Auto
Perform code analysis	AsConfigured
Post-build script arguments	
Post-build script path	
Pre-build script arguments	
Pre-build script path	
3. Test	
1. Automated tests	1 set(s) of tests specified.
2. Advanced	
4. Publish Symbols	
5. Advanced	

Package Restore with Team Foundation Build

For Team Foundation Build 2013 on-premises, the default Build Process Templates already implement the **NuGet Package Restore Workflow** without any special configuration.

To avoid NuGet Package Restore from outside the plugin, you need to remove it from the workflow.

License

The MSBuild Artifactory plugin is available under the **Apache v2 License**.

Screencast

Changelog

[Click here to expand...](#)

2.2.0 (10 Mar 2016)

1. Support for TFS 2015

2.1.0 (18 Aug 2015)

1. Support for Visual Studio 2015
2. Bug fix [MSBAI-7](#)

2.0.0 (30 Apr 2015)

1. Artifactory plugin as a **Visual Studio Extension**, and used as a "Project template" in the solution.
2. Supporting [Black Duck](#) Code Center as part of the build process.

1.0.0 (30 Sep 2014)

1. First release version.

VS Team Services Artifactory Plugin

Overview

Artifactory brings continuous integration to [Visual Studio Team Services\(VSTS\)](#) through the **Visual Studio Team Services Artifactory Plugin**.

Artifactory already provides a set of plugins for [Maven](#), [Gradle](#), [Ivy](#) and other [build tools](#) that are supported on VSTS and enable you to capture information about deployed artifacts, resolve dependencies and deploy artifacts to Artifactory. The Visual Studio Team Services Artifactory plugin adds the ability to deploy and download generic artifacts, promote a build to Artifactory and view build information and promotion history.

Download and Installation

The VSTS Artifactory Plugin is an extension for VSTS and is available for download by account holders from the [VSTS Marketplace](#).

Installation

To install the VSTS Artifactory Plugin, execute the following steps:

- Sign in to your VSTS account and go to the marketplace. You can find the plugin in the **Build and Release** section where it is named **JFrog Artifactory Integration**.
- On the VSTS Artifactory Plugin page, click "Install".
- Select the account to which you want to apply the plugin and confirm installation.

Page Contents

- [Overview](#)
- [Download and Installation](#)
 - [Installation](#)
- [Configuration](#)
 - [Configuring VSTS Components](#)
 - [Configuring an Artifactory Instance](#)
 - [Configuring Your Build](#)
 - [Running Your Build](#)
 - [Promoting Your Build](#)
 - [Automating Release Workflow](#)
 - [JFrog Artifactory Deployer](#)
 - [JFrog Artifactory Downloader](#)
 - [JFrog Artifactory Build Promotion Task](#)

Configuration

Setting up VSTS to work with Artifactory involves two basic steps:

- [Configuring VSTS components](#)
- [Automating release workflow](#) with custom tasks

Configuring VSTS Components

Access Artifactory through ssl and authorize cross-domain request

The components added to VSTS use AJAX to communicate with Artifactory through the REST API. To enable this, you need to configure Artifactory to authorize cross-domain requests from your VSTS.

Also, as VSTS is on HTTPS you must use an HTTPS connection to your Artifactory instance.

Configuring VSTS requires the following basic steps:

- [Configuring an Artifactory instance](#)
- [Configure your build](#)

Configuring an Artifactory Instance

In VSTS, open your team project collection and go to the **Build** tab and select the **Setup JFrog Artifactory** sub-tab.



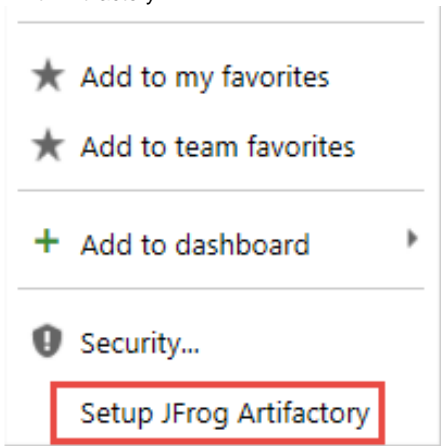
The screenshot shows the VSTS interface with the 'BUILD' tab selected. Below the navigation bar, the 'Setup JFrog Artifactory' sub-tab is active. The main heading is 'Configure JFrog Artifactory'. There are three input fields: 'Artifactory URL' with the value 'https://artifactoryServer/artifactory/', 'User name' with the value 'myuser', and 'Password' which is masked with dots. A 'Save Settings' button is located below the fields.

Provide your **Artifactory URL**, and default login credentials through which VSTS will connect to Artifactory. Once you save your settings they will be stored in VSTS and used for all your projects.

Configuring Your Build

In the **Build Explorer** choose the build definition you want to configure to work with Artifactory and right click on it (in the tree).

The VSTS Artifactory plugin adds an action called **Setup JFrog Artifactory** in the menu which allows you to configure your build definition to work with Artifactory.



Select **Setup JFrog Artifactory** to display the configuration form.

SETUP ARTIFACTORY FOR THIS BUILD

Configure Artifactory

Artifactory URL

Override credentials

User name

Password

Publish repo key

Promote repo key

Once fill in the form and save these settings, they are saved in VSTS as variables which you can use in your build definition files (such as `pom.xml` and other scripts).

Field	Variable	Description
Artifactory URL	ArtifactoryUrl	Your Artifactory server URL
Override credentials	N/A	If checked, you can enter credentials to override the default credentials you provided under Configuring an Artifactory Instance .
User name/ Password	ArtifactoryUsername / ArtifactoryPassword	Artifactory login credentials to use if you have selected Override credentials .
Publish repo key	PublishRepository	The default repository for deployment.
Promote repo key	PromoteRepository	The default repository for promotion

Running Your Build

Configure your project as usual, setting it up to deploy to Artifactory, and run the build.

Once the build has completed, the build summary includes a new JFrog Artifactory section which displays **Artifactory Build Info**.


Queue new build... Download all logs as zip

Build Succeeded

Build 20160122.6
Ran for 55 seconds (Hosted), completed 3 days ago

Summary Timeline Artifacts **JFrog Artifactory** Tests*

Artifactory Build Info



Build agent Generic (v)
Agent VSO (v)
Artifactory Principal tfs
...

History

staged
Repository
Comment the build has been staged
CI User [jonathan.roque@vso.com](#)
Artifactory User tfs

dev
Repository
Comment dev release has been done
CI User [jonathan.roque@vso.com](#)
Artifactory User tfs

Pre-Production
Repository
Comment preProd environnement has been released
CI User [jonathan.roque@vso.com](#)
Artifactory User tfs

The two buttons demarcated in the image above are used for linking directly to the corresponding build information page in Artifactory, and to promote the build.

Promoting Your Build

If you choose to promote your build, VSTS Artifactory Plugin will display a set of fields through which you can configure the [Build Promotion REST API](#) call that will be used to promote the build.

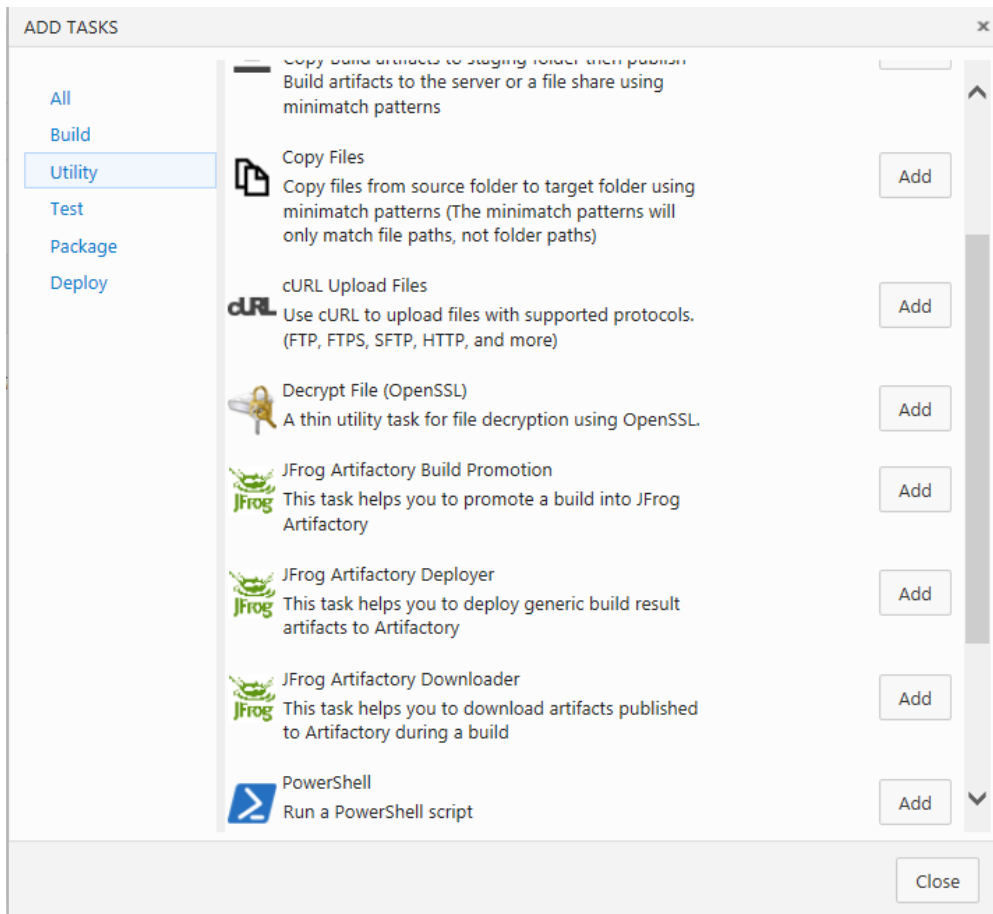
PROMOTE THIS BUILD TO JFROG ARTIFACTORY ✕

Promote to Artifactory

Target Status

Comment

Attach properties










JFrog Artifactory Deployer

The JFrog Artifactory Deployer task uses the [JFrog CLI](#) and facilitates working with VSTS and Artifactory, effectively, on any build tool such as Maven, Gradle and others, for any packaging format such as NuGet, npm and others.

To use it, you need to configure a new service that points to an Artifactory instance as displayed in the following screenshot.

JFrogArtifactoryDeployer

Artifactory URL	ArtifactoryGC	 Manage 
Target Repository	nuget-stage-local/JFrog/MyTestProject/\${Build.BuildNumber}/	
Override Credentials	<input type="checkbox"/>	
User Login		
User Encrypted Password		
Path to JFrog Artifactory Cli exe	\$/TestVSONline/tools/art.exe	 
Path to the Artifacts.	\$(Agent.BuildDirectory)*TestProject*.nupkg	
Properties	qa=dev;stage=test	
Enable build information	<input checked="" type="checkbox"/>	

Control Options

Enabled	<input checked="" type="checkbox"/>
Continue on error	<input type="checkbox"/>
Always run	<input type="checkbox"/>

 Upload artifacts to JFrog Artifactory using JFrog Artifactory cli.

Artifactory URL	Click "Manage" to set the Artifactory instance to which you want to deploy artifacts.
Target Repository	The target repository and path within that repository to which you want to deploy artifacts. Note that you can use environment variables such as $\$(Build.BuildNumber)$.
Override Credentials	When checked, you can override credentials provided by the service.
User Login	User name to use if you have checked Override Credentials .
User Encrypted Password	Corresponding encrypted password to use if you have checked Override Credentials .
Path to JFrog Artifactory Cli.exe	Path to the Artifactory CLI in your VSTS team project code repository.
Path to the Artifacts	Path to the artifacts you want to deploy. You may use environment variables and wild card characters.
Properties	Properties to attach to deployed artifacts.
Enable build information	If checked, build information will be created.

In the example above, we are deploying any NuGet package containing "VSODemo" in its name that is located in the build directory or subfolder of the Artifactory configured as service "ArtifactoryGC" in the repository called "nuget-stage-local" with path "JFrog/MyTestProject/<BuildNumberValue>". Some properties will be attached and the build information provided.

Once the build is complete, you can view its details in Artifactory as displayed below:

Build Browser

History for Build 'DemoVSO'

7 Builds

Filter by Build ID

Build ID	CI Server	Status	Build Time
20160120.4	https://frogdev.visualstudio.com/DefaultCollection/5e470364-4033-447a-862f-cb348cd9764b/_b...		20-01-16 12:53:30 +00:00
20160120.3	https://frogdev.visualstudio.com/DefaultCollection/5e470364-4033-447a-862f-cb348cd9764b/_b...		20-01-16 12:50:01 +00:00
20160120.2	https://frogdev.visualstudio.com/DefaultCollection/5e470364-4033-447a-862f-cb348cd9764b/_b...		20-01-16 12:36:44 +00:00
20160120.1	https://frogdev.visualstudio.com/DefaultCollection/5e470364-4033-447a-862f-cb348cd9764b/_b...		20-01-16 12:25:49 +00:00
20160106.1	https://frogdev.visualstudio.com/DefaultCollection/5e470364-4033-447a-862f-cb348cd9764b/_b...	prod	06-01-16 10:05:51 +00:00
20160105.40	https://frogdev.visualstudio.com/DefaultCollection/5e470364-4033-447a-862f-cb348cd9764b/_b...	staged	05-01-16 17:35:34 +00:00
20160105.38	https://frogdev.visualstudio.com/DefaultCollection/5e470364-4033-447a-862f-cb348cd9764b/_b...		05-11-15 14:40:00 +00:00

You can also view details for each build including a link back to the VSTS build summary:

Build Browser

Build #20160120.4

General Build Info

Published Modules

Environment

Issues

Licenses

Governance

General Info

Name:	DemoVSO
Number:	20160120.4
Agent:	VSO
Build Agent:	Generic
Started:	2016-01-20T12:53:30.300+0000
Duration:	0.0 seconds
Principal:	jon
Artifactory Principal:	tfs
URL:	https://frogdev.visualstudio.com/DefaultCollection/5e470364-4033-447a-862f-cb348cd9764b/_build#buildId=123&a=summary

Build #20160120.4

General Build Info

Published Modules

Environment

Issues

Licenses

Governance

Diff

Release History

Build Info JSON

Environment Variables

35 Variables

Filter by Key

Page 1 of 2

Key	Value
buildInfo.env.AGENT_BUILDDIRECTORY	C:\a\1
buildInfo.env.AGENT_HOMEDIRECTORY	C:\UR\MMS\Services\MmsTaskAgentProvisioner\Tools\agents\default
buildInfo.env.AGENT_ID	1
buildInfo.env.AGENT_JOBNAME	Build
buildInfo.env.AGENT_MACHINENAME	TASKAGENT2-0009
buildInfo.env.AGENT_NAME	Hosted Agent
buildInfo.env.AGENT_ROOTDIRECTORY	C:\a
buildInfo.env.AGENT_WORKFOLDER	C:\a
buildInfo.env.AGENT_WORKINGDIRECTORY	C:\a\SourceRoot\Mapping\2fa6dfe9-bd25-4ace-8bf6-c3d5630e36f9\job-4c8aaf12-fcb2-4b99-9fda-3ad96ab1cefe
buildInfo.env.BUILDCONFIGURATION	debug
buildInfo.env.BUILDPLATFORM	any cpu
buildInfo.env.BUILD_ARTIFACTSTAGINGDIRECTORY	C:\a\1\1a
buildInfo.env.BUILD_BINARIESDIRECTORY	C:\a\1\1b
buildInfo.env.BUILD_BUILDID	123
buildInfo.env.BUILD_BUILDNUMBER	20160120.4
buildInfo.env.BUILD_BUILDURI	vstfs://Build/Build/123

Artifact Name	Type	Repo Path
VSDemo.1.0.0.0.nupkg	nupkg	nuget-prod-local/JFrog/MyTestProject/20160106.1/VSDemo.1.0.0.0.nupkg
VSDemo.DAL.1.0.0.0.nupkg	nupkg	nuget-prod-local/JFrog/MyTestProject/20160106.1/VSDemo.DAL.1.0.0.0.nupkg
VSDemo.Service.1.0.0.0.nupkg	nupkg	nuget-prod-local/JFrog/MyTestProject/20160106.1/VSDemo.Service.1.0.0.0.nupkg

JFrog Artifacts Downloader

The JFrog Artifacts Downloader task downloads artifacts generated in a build from Artifactory as a zip archive and stored as `$env:temp\artifacts.zip`

The main point of this task is to enable download of artifacts in a release workflow and deploy them in a target environment where you can apply acceptance or other tests to them.

JFrog Artifacts Downloader [Link settings](#) [Remove](#)

Version 1.* ▼

Display name

Artifactory URL ⓘ *
 ▼ ↻ ⚙

Override Credentials ⓘ

User Login ⓘ

User Encrypted Password ⓘ

Build Name. ⓘ *

Build Number ⓘ

Build Status ⓘ

Destination folder ⓘ















Control Options ▼

Artifactory URL	Please refer to parameters for the JFrog Artifacts Downloader task.
Override Credentials	Please refer to parameters for the JFrog Artifacts Downloader task.
User Login	Please refer to parameters for the JFrog Artifacts Downloader task.
User API Key	Please refer to parameters for the JFrog Artifacts Downloader task.
Build Name	The name of the build from which to download artifacts.

Build Number (optional)	If not provided, the task will use the current build number from the BUILD_BUILDNUMBER environment variable. If this variable is not available in context, the task will use the latest build from Artifactory.
Build Status (optional)	If provided, the task only downloads artifacts linked to the latest build with the specified status. If not provided, the task downloads artifacts from the current build number (as determined from the environment variables). If the task is applied in a release workflow (rather than being triggered by a build, and therefore, there is no build number available), the latest build will be used.

JFrog Artifactory Build Promotion Task

JFrogArtifactoryPromote

Artifactory URL	ArtifactoryGC 	 Manage 
Override Credentials	<input type="checkbox"/>	
User Login	<input type="text"/>	
User Encrypted Password	<input type="text"/>	
Build Name.	<input type="text"/>	
Build number	<input type="text"/>	
Status	dev	
Comment	dev release has been done	
Target repository	<input type="text"/>	
Copy artifacts	<input type="checkbox"/>	
Copy/move build's dependencies	<input type="checkbox"/>	
Properties	<input type="text"/>	

Control Options

- Enabled
- Continue on error
- Always run

 Promote a build into Artifactory.

Artifactory URL	Please refer to parameters for the JFrog Artifactory Deployer task.
Target Repository	Please refer to parameters for the JFrog Artifactory Deployer task.
Override Credentials	Please refer to parameters for the JFrog Artifactory Deployer task.
User Login	Please refer to parameters for the JFrog Artifactory Deployer task.

User API Key	Please refer to parameters for the JFrog Artifactory Deployer task.
Build Name (optional)	The name of the build from which to download artifacts. If not provided, the task will use values from environment variables if available.
Build Number (optional)	The number of the build from which to download artifacts. If not provided, the task will use values from environment variables if available.
Build Status	Target status for promotion.
Comment (Optional)	Optional comment to add.
Target Repository (Optional)	You may set this parameter if you want the build moved/copied to a target repository.
Copy artifacts	If checked, build artifacts will be copied, otherwise they will be moved.
Copy/move build dependencies	If checked, the build dependencies will also be copied/moved in the promotion.
Properties (optional)	You may add a set of properties separated by a semicolon.

Using File Specs

Overview

File Specs can be used to specify the details of files you want to upload or download to or from Artifactory. They are specified in JSON format and are currently supported in [JFrog CLI](#), [Jenkins Artifactory Plugin](#), [TeamCity Artifactory Plugin](#) and [Bamboo Artifactory Plugin](#).

This article describes the File Specs structure used in Jenkins, TeamCity and the Bamboo Artifactory plugins.

You can read about JFrog CLI with File Specs [here](#).

Download Spec Schema

The download spec schema offers the option of using [AQL](#) or wildcard patterns according to the JSON element you specify:

Page contents

- [Overview](#)
- [Download Spec Schema](#)
- [Upload Spec Schema](#)
- [Using Placeholders](#)
- [Examples](#)


```

{
  "files": [
    {
      "pattern" or "aql": "[Mandatory]",
      "target": "[Mandatory]",
      "props": "[Optional]",
      "recursive": "[Optional, Default: true]",
      "flat": "[Optional, Default: false]",
      "build": "[Optional]",
      "explode": "[Optional, Default: false]",
      "excludePatterns": ["[Optional]"]
    }
  ]
}

```

Where:

Element	Description
pattern	[Mandatory if 'aql' is not specified] Specifies the source path in Artifactory, from which the artifacts should be downloaded, in the following format: [repository name]/[repository path]. You can use wildcards to specify multiple artifacts.
target	[Mandatory] Specifies the local file system path to which artifacts which should be downloaded. For flexibility in specifying the target path, you can include placeholders in the form of {1}, {2}, {3}...which are replaced by corresponding tokens in the pattern property that are enclosed in parenthesis. For more details, please refer to Using Placeholders . Since version 2.9.0 of the Jenkins Artifactory plugin and version 2.3.1 of the TeamCity Artifactory plugin the target format has been simplified and uses the same file separator "/" for all operating systems, including Windows.
aql	[Mandatory if 'pattern' is not specified] An AQL query that specified the artifacts to be downloaded.
props	[Optional] List of "key=value" pairs separated by a semi-colon. (For example, "key1=value1;key2=value2;key3=value3"). Only artifacts with all of the specified properties and values will be downloaded.
flat	[Default: false] If true, artifacts are downloaded to the exact target path specified and their hierarchy in the source repository is ignored. If false, artifacts are uploaded to the target path in the file system while maintaining their hierarchy in the source repository.
recursive	[Default: true] If true, artifacts are also downloaded from sub-paths under the specified path in the source repository. If false, only artifacts in the specified source path directory are downloaded.
build	[Optional] If specified, only artifacts of the specified build are downloaded. The 'pattern' property is still taken into account when 'build' is specified. The property format is build-name/build-number. If the build number is not specified, or the keyword LATEST is used for the build number, then the latest published build number is used.
explode	[Default: false] If true, the downloaded archive file is extracted after the download. The archived file itself is deleted locally. The supported archive types are: zip, tar; tar.gz; and tgz

excludePatterns	<p>[Optional. Applicable only when 'pattern' is specified]</p> <p>Note: excludePatterns are not yet supported in Bamboo.</p> <p>An array (enclosed with square brackets) of patterns to be excluded from downloading. Unlike the "pattern" property, "excludePatterns" must NOT include the repository as part of the pattern's path. You can use wildcards to specify multiple artifacts.</p> <p>For example: ["*.sha1","*.md5"]</p>
------------------------	---

Upload Spec Schema

```

{
  "files": [
    {
      "pattern": "[Mandatory]",
      "target": "[Mandatory]",
      "props": "[Optional]",
      "recursive": "[Optional, Default: 'true']",
      "flat": "[Optional, Default: 'true']",
      "regexp": "[Optional, Default: 'false']",
      "explode": "[Optional, Default: false]",
      "excludePatterns": "[Optional]"
    }
  ]
}

```

Where:

Element	Description
pattern	<p>[Mandatory]</p> <p>Specifies the local file system path to artifacts which should be uploaded to Artifactory. You can specify multiple artifacts by using wildcards or a regular expression as designated by the regexp property. If you use a regexp, you need to escape any reserved characters (such as ".", "?", etc.) used in the expression using a backslash "\".</p> <p>Since version 2.9.0 of the Jenkins Artifactory plugin and version 2.3.1 of the TeamCity Artifactory plugin the pattern format has been simplified and uses the same file separator "/" for all operating systems, including Windows.</p>
target	<p>[Mandatory]</p> <p>Specifies the target path in Artifactory in the following format: [repository_name]/[repository_path]</p> <p>If the pattern ends with a slash, for example "repo-name/a/b/", then "b" is assumed to be a folder in Artifactory and the files are uploaded into it. In the case of "repo-name/a/b", the uploaded file is renamed to "b" in Artifactory.</p> <p>For flexibility in specifying the upload path, you can include placeholders in the form of {1}, {2}, {3}...which are replaced by corresponding tokens in the source path that are enclosed in parenthesis. For more details, please refer to Using Placeholders.</p>
props	<p>[Optional]</p> <p>List of "key=value" pairs separated by a semi-colon (;) to be attached as properties to the uploaded properties. If any key can take several values, then each value is separated by a comma (,). For example, "key1=value1;key2=value21,value22;key3=value3".</p>
flat	<p>[Default: true]</p> <p>If true, artifacts are uploaded to the exact target path specified and their hierarchy in the source file system is ignored. If false, artifacts are uploaded to the target path while maintaining their file system hierarchy.</p>

recursive	[Default: true] If true, artifacts are also collected from sub-directories of the source directory for upload. If false, only artifacts specifically in the source directory are uploaded.
regexp	[Default: false] If true, the command will interpret the pattern property, which describes the local file-system path of artifacts to upload, as a regular expression. If false, the command will interpret the pattern property as a wild-card expression.
explode	[Default: false] If true, the uploaded archive file is extracted after it is uploaded. The archived file itself is not saved in Artifactory. The supported archive types are: zip, tar; tar.gz; and tgz
excludePatterns	[Optional] Note: excludePatterns are not yet supported in Bamboo. An array (enclosed with square brackets) of patterns to be excluded from uploading. Allows using wildcards or a regular expression as designated by the regexp property. If you use a regexp, you need to escape any reserved characters (such as ".", "?", etc.) used in the expression using a backslash "\". For example: ["*.sha1","*.md5"]

Using Placeholders

File Specs offer enormous flexibility in how you **upload**, or **download** files through use of wildcard or regular expressions with placeholders.

Any wildcard enclosed in parenthesis in the pattern property can be matched with a corresponding placeholder in the target property to determine the name of the artifact once downloaded or uploaded. The following example downloads all zip files, located under the root path of the *my-local-repo* repository, which include a dash in their name. The files are renamed when they are downloaded, replacing the dash with two dashes.

```
{
  "files": [
    {
      "pattern": "my-local-repo/(*)-(*) .zip",
      "target": "froggy/{1}--{2}.zip",
      "recursive": "false"
    }
  ]
}
```

Examples

Example 1: Download all files located under the *all-my-frogs* directory in the *my-local-repo* repository to the *froggy/all-my-frogs* directory.

```
{
  "files": [
    {
      "pattern": "my-local-repo/all-my-frogs/",
      "target": "froggy/"
    }
  ]
}
```

Example 2: Download all files retrieved by the AQL query to the *froggy* directory.

```
{
  "files": [
    {
      "aql": {
        "items.find": {
          "repo": "my-local-repo",
          "$or": [
            {
              "$and": [
                {
                  "path": {
                    "$match": "."
                  },
                  "name": {
                    "$match": "al.in"
                  }
                }
              ]
            },
            {
              "$and": [
                {
                  "path": {
                    "$match": "*"
                  },
                  "name": {
                    "$match": "al.in"
                  }
                }
              ]
            }
          ]
        }
      },
      "target": "cli-reg-test/spec-copy-test/aql-al/"
    }
  ]
}
```

Example 3: Upload

1. All zip files located under the *resources* directory to the *zip* folder, under the *all-my-frogs* repository.
- AND
2. All TGZ files located under the *resources* directory to the *tgz* folder, under the *all-my-frogs* repository.
3. Tag all zip files with type = zip and status = ready.
4. Tag all tgz files with type = tgz and status = ready.

```

{
  "files": [
    {
      "pattern": "resources/*.zip",
      "target": "my-repo/zip/",
      "props": "type=zip;status=ready"
    },
    {
      "pattern": "resources/*.tgz",
      "target": "my-repo/tgz/",
      "props": "type=tgz;status=ready"
    }
  ]
}

```

Example 4: Download all files located under the *all-my-frogs* directory in the *my-local-repo* repository **except** for the '.zip' files and the 'props.' files

```

{
  "files": [
    {
      "pattern": "my-local-repo/all-my-frogs/",
      "excludePatterns": [
        "*.zip",
        "all-my-frogs/props.*"
      ]
    }
  ]
}

```

Troubleshooting

Overview

The Artifactory User Guide provides troubleshooting tips for different topics on the relevant pages describing those topics.

Installing Artifactory

Please refer to Troubleshooting under [Installing Artifactory](#)

Installing with Docker

Please refer to Troubleshooting Docker under [Installing with Docker](#)

Installing Artifactory HA

Please refer to [Troubleshooting HA](#).

Access Tokens or Access Service

Please refer to Troubleshooting under [Access Tokens](#).

Known Issues

The following table lists the set of known issues in Artifactory including the version in which they were discovered, and the version in which they were fixed. Click the issue ID for full details in JIRA.

Issue ID	Description	Affected From Version	Fix Version
RTFACT-15315	Running <code>apt-get update</code> on Ubuntu Trusty (14.04) against Debian repositories fails with the following error: <code>Sub-process https received a segmentation fault</code>	5.6.0	5.6.1
RTFACT-15297	For Artifactory HA installations, single-phase upgrades (with downtime) from version 4.x to version 5.6 without going through version 5.4.6 fails. Please refer to the Upgrade Notice in the Artifactory 5.5 Release Notes .	5.6.0	5.6.1
RTFACT-14672	Git LFS client v1.x working against Artifactory Git LFS repository using SSH fails to upload or download LFS blobs.	5.3.0	5.4.6
RTFACT-14473	When resolving a package from an npm repository, Artifactory throws a deserialize error to the log file if one of the package's dependencies in the corresponding <code>package.json</code> file is declared using the following format: " <code><dependency_name></code> ": { "version": " <code><version_number></code> " }, For example: "deep-diff" uses this format. As a result, the npm client fails to resolve the package. Note: For a workaround, please refer to the issue details .	5.4.0	5.4.6
RTFACT-14687	A system import on a High Availability installation of Artifactory 5.4.x with files exported from a version that is below 5.4, will fail.	5.4.0	5.6.0
RTFACT-14530	After upgrading an Artifactory HA cluster from version 5.x to 5.4.x, new nodes that you add to your Artifactory HA cluster will not start up.	5.4.0	5.4.4
RTFACT-14510	Uploading or downloading files to Artifactory using access tokens with a subject that is longer than 64 characters fails with error 500.	5.4.2	5.4.3
RTFACT-14477	Artifactory fails to start up when Tomcat is configured to only use HTTPS, or was configured with both HTTP and HTTPS but on different ports.	5.4.0	5.4.2
RTFACT-14495	Artifactory is unable to connect to access service and as a result cannot start when Tomcat is configured with a Self-Signed chain certificate.	5.4.0	5.4.2
RTFACT-14279	Following an upgrade from Artifactory version 4.4 or below to 5.3 directly, Artifactory will fail to start up. 23-May-2017: If upgrading from 4.4 or below , we recommend waiting for a patch which should be released shortly. Note: Upgrading from version 4.4.1 and above to 5.3 is not affected and Artifactory will start up after the upgrade.	5.3.0	5.3.1
RTFACT-14063	An external user who has created a token will still be able to refresh it even if he has been removed from the external authentication server.	5.2.1	
RTFACT-13870	Deploying artifacts larger than 100MB in size when using an S3 compatible storage provider can fail leading to unpredictable results.	5.1.2	5.1.3
RTFACT-13823	JFrog Mission Control reload plugin may get into a loop. Note that as a workaround this plugin can be removed since it is no longer used if working with Artifactory version 5.0.0 or above.	5.1.0	
RTFACT-14619	User Plugins that contain the <code>realms</code> execution point will fail to be executed and throw an <code> HazelcastSerializationException </code> exception when running on Artifactory HA installations.	5.0.0	
RTFACT-13923	Artifactory might not start up following an upgrade to version 5.x on Windows when Artifactory is configured with a Keystore .	5.0.0	5.1.4
RTFACT-13822	Most downloaded widget in Artifactory home can cause the DB to stall.	5.0.0	5.1.3

RTFACT-14079	When using FullDB or S3 as a binary provider and an NPM virtual repository, external dependencies cause a leak of resources that exhausts all of the DB connections that are never freed.	4.16.0	5.3.0
RTFACT-13764	"downloadTagFile" and "downloadBranchFile" API endpoints throw a nullPointerException when a VCS remote repository has "Store Artifacts locally" disabled	4.16.0	
RTFACT-8503	When viewing Smart Remote Repositories in the tree browser, folders under the repository that contain a space in their names, will not expand.	4.0.0	
RTFACT-8194	'docker pull' fails on remote repositories when 'Store Artifacts Locally' is disabled.	3.9.0	
RTFACT-11942	'npm install' fails when all of the following conditions occur together: <ul style="list-style-type: none"> You are trying to resolve a scoped package from a virtual repository and, The scoped package is actually hosted under one of the remote repositories aggregated by the virtual repository and, That remote repository has 'Store artifacts locally' disabled. 	3.4.2	
RTFACT-13004	When trying to retrieve an item's properties from Docker V2 repositories using the /api/storage REST API endpoint, a 401 error is returned to the client.	3.4.2	
RTFACT-12710	Python metadata calculation will fail if the Python metadata version is set to 1.2 inside the METADATA or PKG-INFO files.	3.4.0	5.4.0
RTFACT-12379	NuGet virtual repositories that aggregate more than one local or remote repository may omit results when searching for a package	2.5.0	5.2.1
RTFACT-10132	'NuGet install' does not enforce include / exclude patterns on virtual NuGet repositories	2.5.0	5.7.0
RTFACT-13618	When all artifacts have been deleted from a folder, the folder is not pushed to the pruning queue even though it is empty, and is, therefore, not deleted.	2.0.0	
RTFACT-12260	When performing a full system export on an instance with blacked out repositories, and then doing a corresponding import on a target instance, the content and metadata of the blacked out repositories are not imported.	2.0.0	
RTFACT-12934	A Maven virtual repository does not return the correct latest snapshot if another package with the same version number exists in a different repository.	2.0.0	
RTFACT-12959	Anonymous users cannot download folders even if anonymous access is enabled.	2.0.0	5.6.0
RTFACT-6485	Uploading a file containing a dot in the target path will fail and a 500 error will be returned to the client Note: The <code>maven-site-plugin</code> creates upload URLs containing a dot. Therefore, uploading artifacts through the plugin is currently not supported	2.0.0	5.4.0
RTFACT-9343	Promoting Build Info that contains a dependency or an artifact that has the same checksum of an item that already exists may result in a race condition for the same target path.	2.0.0	

End of Life

JFrog supports all versions of Artifactory from their date of release going forward 18 months.

Here is the list of versions and their End of Life (EoL) date:

Version	Release Date	EoL Date
5.6.0	15-Nov-2017	15-May-2019
5.5.2	29-Oct-2017	29-Apr-2019
5.5.1	26-Sep-2017	26-Mar-2019
5.5.0	25-Sep-2017	25-Mar-2019
5.4.5	18-Jul-2017	18-Jan-2019
5.4.4	06-Jul-2017	06-Jan-2019

5.4.3	03-Jul-2017	03-Jan-2019
5.4.2	30-Jun-2017	30-Dec-2018
5.4.1	22-Jun-2017	22-Dec-2018
5.4.0	20-Jun-2017	20-Dec-2018
5.3.1	24-May-2017	24-Nov-2018
5.3.0	11-May-2017	11-Nov-2018
5.2.1	13-April-2017	13-Oct-2018
5.2.0	27-Mar-2017	27-Sep-2018
5.1.4	19-Mar-2017	19-Sep-2018
5.1.3	09-Mar-2017	09-Sep-2018
5.1.0	21-Feb-2017	21-Aug-2018
5.0.1	07-Feb-2017	07-Aug-2018
5.0.0	31-Jan-2017	31-Jul-2018
4.16.0	16-Jan-2017	16-Jul-2018
4.15.0	15-Dec-2016	15-Jun-2018
4.14.3	07-Dec-2016	07-Jun-2018
4.14.2	27-Nov-2016	27-May-2018
4.14.1	01-Nov-2016	01-May-2018
4.14.0	20-Oct-2016	20-Apr-2018
4.13.1	13-Oct-2016	13-Apr-2018
4.13.0	21-Sep-2016	21-Mar-2018
4.12.2	14-Sep-2016	14-Mar-2018
4.12.1	07-Sep-2016	07-Mar-2018
4.12.0.1	29-Aug-2016	01-Mar-2018
4.11.2	17-Aug-2016	17-Feb-2018
4.11.1	14-Aug-2016	14-Feb-2018
4.11.0	01-Aug-2016	01-Feb-2018
4.10.0	20-Jul-2016	20-Jan-2018
4.9.1	13-Jul-2016	13-Jan-2018
4.9.0	01-Jul-2016	01-Jan-2018
4.8.2	19-Jun-2016	19-Dec-2017
4.8.1	09-Jun-2016	09-Dec-2017
4.8.0	23-May-2016	23-Nov-2017
4.7.7	14-May-2016	14-Nov-2017
4.7.6	09-May-2016	09-Nov-2017
4.7.5	28-Apr-2016	28-Oct-2017
4.7.4	19-Apr-2016	19-Oct-2017

4.7.3	17-Apr-2016	17-Oct-2017
4.7.2	14-Apr-2016	14-Oct-2017
4.7.1	04-Apr-2016	04-Oct-2017
4.7.0	31-Mar-2016	30-Sep-2017
4.6.1	21-Mar-2016	21-Sep-2017
4.6.0	13-Mar-2016	13-Sep-2017
4.5.2	28-Feb-2016	28-Aug-2017
4.5.1	18-Feb-2016	18-Aug-2017
4.5.0	14-Feb-2016	14-Aug-2017
4.4.3	08-Feb-2016	08-Aug-2017
4.4.2	17-Jan-2016	17-Jul-2017
4.4.1	12-Jan-2016	12-Jul-2017
4.4.0	04-Jan-2016	04-Jul-2017
4.3.3	21-Dec-2015	21-Jun-2017
4.3.2	08-Dec-2015	08-Jun-2017
4.3.1	03-Dec-2015	03-Jun-2017
4.3.0	22-Nov-2015	22-May-2017
4.2.2	05-Nov-2015	05-May-2017
4.2.1	01-Nov-2015	01-May-2017
4.2.0	18-Oct-2015	18-Apr-2017
4.1.3	27-Sep-2015	27-Mar-2017
4.1.2	20-Sep-2015	20-Mar-2017
4.1.0	08-Sep-2015	08-Mar-2017
4.0.2	12-Aug-2015	12-Feb-2017
4.0.1	09-Aug-2015	09-Feb-2017
4.0.0	02-Aug-2015	02-Feb-2017
3.9.5	29-Oct-2015	13-Feb-2017
3.9.4	13-Aug-2015	13-Feb-2017
3.9.3	09-Aug-2015	09-Feb-2017
3.9.2	06-Jul-2015	05-Jan-2017
3.9.1	24-Jun-2015	24-Dec-2016
3.9.0	21-Jun-2015	21-Dec-2016
3.8.0	31-May-2015	21-Nov-2016
3.7.0	17-May-2015	17-Nov-2016
3.6.0	12-Apr-2015	12-Oct-2016
3.5.3	22-Mar-2015	22-Sep-2016
3.5.2.1	24-Feb-2015	24-Jun-2016

3.5.2	23-Feb-2015	23-Jun-2016
3.5.1	04-Feb-2015	04-Jun-2016
3.5.0	01-Feb-2014	01-Jun-2016
3.4.2	30-Nov-2014	30-May-2016
3.4.1	21-Oct-2014	21-Apr-2016
3.4.0.1	06-Oct-2014	06-Apr-2016
3.4.0	30-Sep-2014	30-Mar-2016
3.3.1	09-Sep-2014	09-Mar-2016
3.3.0.1	10-Aug-2014	10-Feb-2016
3.3.0	13-Jul-2014	13-Jan-2016
3.2.2	22-Jun-2014	22-Dec-2015
3.2.1.1	11-Feb-2014	11-Aug-2016
3.2.1	01-Jun-2014	01-Dec-2015
3.2.0	30-Mar-2014	30-Sep-2015
3.1.1.1	11-Feb-2014	11-Aug-2015
3.1.1	09-Feb-2014	09-Aug-2015
3.1.0	15-Dec-2013	15-Jun-2015
3.0.4	27-Oct-2013	27-Apr-2015
3.0.3	11-Aug-2013	11-Feb-2015
3.0.2	08-Jul-2013	08-Jan-2015
3.0.1	13-May-2013	13-Nov-2014
3.0.0	21-Apr-2013	21-Oct-2014
2.6.7	27-Jun-2013	27-Dec-2014
2.6.6	20-Dec-2012	20-Jun-2014
2.6.5	12-Nov-2012	12-May-2014
2.6.4	20-Oct-2012	20-Apr-2014
2.6.3	02-Aug-2012	02-Feb-2014
2.6.2	23-Jul-2012	23-Jan-2014
2.6.1	23-May-2012	23-Nov-2013
2.6.0	06-May-2012	05-Nov-2013
2.5.2	25-Apr-2012	25-Oct-2013
2.5.1.1	06-Mar-2012	06-Sep-2013
2.5.1	15-Feb-2012	15-Aug-2013
2.5.0	31-Jan-2012	31-Jul-2013
2.4.2	29-Nov-2011	29-May-2013
2.4.1	16-Nov-2011	16-May-2013
2.4.0	08-Nov-2011	08-May-2013

2.3.2	03-Jan-2012	03-Jul-2013
2.3.1	14-Feb-2011	14-Aug-2012
2.3.0	20-Oct-2010	20-Apr-2012
2.2.3	25-Apr-2010	25-Oct-2011
2.2.2	22-Mar-2010	22-Sep-2011
2.2.1	17-Feb-2010	17-Aug-2011
2.2.0	08-Feb-2010	08-Aug-2011
2.1.3	21-Dec-2009	21-Jun-2011
2.1.2	03-Nov-2009	03-May-2011
2.1.1	21-Oct-2009	21-Apr-2011
2.1.0	05-Oct-2009	05-Apr-2011
2.0.8	08-Sep-2009	08-Mar-2011
2.0.7	12-Aug-2009	12-Feb-2011
2.0.6	14-May-2009	14-Nov-2010
2.0.5	08-Apr-2009	08-Oct-2010
2.0.4	31-Mar-2009	31-Sep-2010
2.0.3	17-Mar-2009	17-Sep-2010
2.0.2	18-Feb-2009	18-Aug-2010
2.0.1	09-Feb-2009	09-Aug-2010
2.0.0	05-Jan-2009	05-Jul-2010

This table is updated periodically please excuse any delay in updates. You can check for a specific version by checking the version release date and verify the release date is within the last 18 months.

Release Notes

Overview

This page presents release notes for JFrog Artifactory describing the main fixes and enhancements made to each version as it is released. For a complete list of changes in each version, please refer to the **JIRA Release Notes** linked at the end of the details for each release.

If you need release notes for earlier versions of Artifactory, please refer to the [Release Notes](#) in the Artifactory 3.x User Guide.

Download

For an Artifactory Pro or Artifactory Enterprise installation, click to download the latest version of **JFrog Artifactory Pro**.

For an Artifactory OSS installation, click to download the latest version of **JFrog Artifactory OSS**.

Previous Versions

Previous versions of JFrog Artifactory Pro and JFrog Artifactory OSS are available for download on JFrog Bintray.

Click to download previous versions of **JFrog Artifactory Pro**.

Click to download previous versions of JFrog Artifactory OSS as a **ZIP** or **RPM**.

Page Contents

- [Overview](#)
 - [Download](#)
 - [Previous Versions](#)
 - [Upgrade Notice](#)
 - [Installation and Upgrade](#)
 - [Known Issues](#)
- [Artifactory 5.6](#)
 - [Artifactory 5.6.1](#)
 - [Artifactory 5.6.2](#)
- [Artifactory 5.5](#)
 - [Artifactory 5.5.1](#)
 - [Artifactory 5.5.2](#)
- [Artifactory 5.4](#)
 - [Artifactory 5.4.1](#)
 - [Artifactory 5.4.2](#)
 - [Artifactory 5.4.3](#)
 - [Artifactory 5.4.4](#)
 - [Artifactory 5.4.5](#)
 - [Artifactory 5.4.6](#)
- [Artifactory 5.3](#)
 - [Artifactory 5.3.1](#)
 - [Artifactory 5.3.2](#)
- [Artifactory 5.2](#)
 - [Artifactory 5.2.1](#)

Upgrade Notice

Artifactory 5.5 implements a database schema change to natively support SHA-256 checksums. This change affects the upgrade procedure for an Enterprise Artifactory HA cluster (upgrading an Artifactory Pro or OSS installation is not affected).

For an Artifactory Enterprise HA cluster, **if your current version is 5.4.6**, you may proceed with the normal upgrade procedure described in [Upgrading an Enterprise HA Cluster](#).

If your current version is below 5.4.6, there are two options to upgrade to the latest version (5.5 and above): a two-phase option with zero downtime or a single phase option that incurs downtime.

For details, please refer to the [Upgrade Notice](#) under the release notes for [Artifactory 5.5.1](#).

Longer upgrade time

Due to the changes implemented in version 5.5, upgrading to this version or above from version 5.4.6 or below may take longer than usual and depends on the database you are using.

For an Artifactory Pro installation and for the Primary node of an Artifactory HA cluster, if you use MySQL database, the upgrade may take up to 5 minutes for each 1 million artifacts in your repositories for a typical setup. If you are using one of the other supported databases, the extra upgrade time will be less noticeable and should only take several seconds longer than usual.

Installation and Upgrade

For installation instructions please refer to [Installing Artifactory](#).

To upgrade to this release from your current installation please refer to [Upgrading Artifactory](#).



To receive automatic notifications whenever there is a new release of Artifactory, please watch us on [Bintray](#).

Known Issues

For a list of known issues in the different versions of Artifactory, please refer to [Known Issues](#).

Artifactory 5.6

Released: November 15, 2017

Upgrade Notice

Before Upgrading to Artifactory 5.6.0

1. The [Artifactory Security Replication User Plugin](#) (`securityReplication.groovy`) has not yet been updated to support 5.6.0. We're working on a new version that will be **available soon**.

If you are using this plugin and need to upgrade to Artifactory 5.6.0, please contact support@jfrog.com.

2. For Artifactory HA installations, [single-phase upgrades](#) (with downtime) from version 4.x to version 5.6 without going through version 5.4.6 fails. Please refer to the Upgrade Notice in the [Artifactory 5.5 Release Notes](#).
3. There is a known issue in which running `apt-get update` on Ubuntu Trusty (14.04) against Debian repositories fails with the following error: `Sub-process https received a segmentation fault`

A fix for this issue is available in version 5.6.1 and we therefore recommend upgrading to 5.6.1.

- Artifactory 5.1
 - Artifactory 5.1.2
 - Artifactory 5.1.3
 - Artifactory 5.1.4
- Artifactory 5.0
 - Artifactory 5.0.1
- Artifactory 4.16
 - Artifactory 4.16.1
- Artifactory 4.15
- Artifactory 4.14
 - Artifactory 4.14.1
 - Artifactory 4.14.2
 - Artifactory 4.14.3
- Artifactory 4.13
 - Artifactory 4.13.1
 - Artifactory 4.13.2
- Artifactory 4.12.0.1
 - Artifactory 4.12.1
 - Artifactory 4.12.2
- Artifactory 4.11
 - Artifactory 4.11.1
 - Artifactory 4.11.2
- Artifactory 4.10
- Artifactory 4.9
 - Artifactory 4.9.1
- Artifactory 4.8
 - Artifactory 4.8.1
 - Artifactory 4.8.2
- Artifactory 4.7
 - Artifactory 4.7.1
 - Artifactory 4.7.2
 - Artifactory 4.7.3
 - Artifactory 4.7.4
 - Artifactory 4.7.5
 - Artifactory 4.7.6
 - Artifactory 4.7.7
- Artifactory 4.6
 - Artifactory 4.6.1
- Artifactory 4.5
 - Artifactory 4.5.1
 - Artifactory 4.5.2
- Artifactory 4.4
 - Artifactory 4.4.1
 - Artifactory 4.4.2
 - Artifactory 4.4.3
- Artifactory 4.3
 - Artifactory 4.3.1
 - Artifactory 4.3.2
 - Artifactory 4.3.3
- Artifactory 4.2
 - Artifactory 4.2.1
 - Artifactory 4.2.2
- Artifactory 4.1
 - Artifactory 4.1.2
 - Artifactory 4.1.3
- Artifactory 4.0
 - Artifactory 4.0.1
 - Artifactory 4.0.2
- Previous Release Notes

Highlights

Improved Debian Performance

Significant improvement in performance when indexing Debian repositories.

Feature Enhancements

Tomcat Version Upgrade

The Tomcat bundled with Artifactory has been upgraded to version 8.5.23.

Get Distribution Repository Details

The [Get Repositories](#) REST API now also includes [distribution repositories](#). To get the distribution repositories details only, you can add `type=distribution` as a query param.

UI Performance Improvement

Performance of displaying the environment and system variables data in the Builds module in the UI has been significantly improved.

Downloading a Folder for Anonymous Users

Admin users can now also enable [folder download configuration](#) for [anonymous users](#), in addition to internal users.

Limit REST API Search Results

Added the ability to limit the number of API search results for internal users, previously available only for anonymous users. To add a limit, edit the [artifactory.system.properties](#) file with `artifactory.search.limitAnonymousUsersOnly=false` (default is true), and a limit `artifactory.search.userQueryLimit` (default is 1000).

Applicable to the following REST API calls

[Artifact Search](#), [Archive Entries Search](#), [GAVC Search](#), [Property Search](#), [Checksum Search](#) (limited by UI max results), [Artifacts Not Downloaded Since](#), [Artifacts With Date in Date Range](#), [Artifacts Created in Date Range](#).

Filter Expirable Access Tokens

Added an option to filter the expirable tokens in the [Access Tokens](#) page in the Artifactory UI.

Issues Resolved

1. Fixed an issue allowing unsupported special characters to be used in the key field when adding properties via REST API, as already enforced in the UI.
The following characters are forbidden: `(){}[*+^$\\/~`!@#%&<>|=,±§` and the space character.
2. Fixed an issue where a file with the same filename and filepath of a file that was previously deleted, could not be deleted a second time. For this scenario, the latest file deleted will now be under the file path in the trash.
3. Fixed an issue where NuGet package names containing a hyphen character "-" would be automatically considered as pre-release packages which allowed users without Delete/Overwrite permissions to overwrite them.
For example: `Sample-Package.1.0.0.nupkg`
Artifactory is now aligned with the NuGet spec, and these packages will only be considered as pre-release if the hyphen character follows the version number.
For example: `Sample-Package.1.0.0-RC.nupkg`
4. Fixed an issue where installing an npm package, with the following date format (2010-11-09T23:36:08Z) in its metadata file, would fail with an `IllegalArgumentException`.
5. Fixed an issue in which installing an npm package from a virtual repository would fail if the package did not have the `time` closure in the `package.json`.

6. Fixed an issue in which users imported from [Crowd](#) and associated to a group with admin privileges would be created in Artifactory with the "Can Update Profile" option disabled. This option will now be enabled for this usecase.
7. Fixed an issue in which users associated to a group imported from [SAML](#) and associated with admin privileges were not granted the appropriate admin privileges.
8. Fixed an issue where uploading a [Conan package](#) that contains declared environment variables with the "=" character, the package would be deployed without its metadata.
For Example: `conan install lib/1.0@user/stable -e MYFLAG="one==tricky==value" --build`

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.6.1

Released: November 22, 2017

Issues Resolved

1. Fixed an issue in which a [single-phase upgrade](#) of an HA cluster with downtime (by adding the `artifactory.upgrade.allowAnyUpgrade.forVersion` system property) from a version below 5.0 directly to version 5.6.0 would fail. Note that the recommended [two-phase upgrade](#) with zero downtime was not affected.
2. Fixed an issue in which when logging into Artifactory, if the group name sent in a SAML assertion as a SAML attribute was in mixed-case (i.e., at least one character is not lower-case), and the corresponding group in Artifactory was all in lower case, then the SAML user would not inherit the permissions associated with that group. This affected both internal groups and imported LDAP groups.
3. Fixed an issue in which running `apt-get update` on Ubuntu Trusty (14.04) against Debian repositories would fail with the following error:
`Sub-process https received a segmentation fault`

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.6.2

Released: November 27, 2017

Issues Resolved

1. Fixed a critical issue in which a user would sometimes lose permissions due to a collision between an update action and a "GET" operation that occurred concurrently.
2. Fixed an issue that prevented connection to Artifactory through SSH. This also resulted in JFrog CLI not being able to work with Artifactory.

For a complete list of changes please refer to our [JIRA Release Notes](#).

For an Artifactory Pro or Artifactory Enterprise installation, click to download this latest version of [JFrog Artifactory Pro](#).

For Artifactory OSS, click to download this latest version of [JFrog Artifactory OSS](#).

Artifactory 5.5

Released: September 25, 2017

Due to a critical issue discovered in this version, you should not install it. Instead, you should upgrade to version 5.5.1 or later.

Upgrade Notice

**For an Artifactory HA installation, there are two options to upgrade to version 5.5 from a version below 5.4.6
This note only refers to upgrading Artifactory Enterprise HA installations.**

Artifactory 5.5 implements a database schema change to natively support SHA-256 checksums. **If your current version is 5.4.6**, you may proceed with the normal upgrade procedure described in [Upgrading an Enterprise HA Cluster](#).

If your current version is below 5.4.6, to accommodate this change, you may select one of the following two upgrade options:

1. **Two-phase, zero downtime**

In this option, you first need to upgrade your HA cluster to version 5.4.6. Once this upgrade is completed, you can then proceed to upgrade your HA cluster to version 5.5. In both phases, you follow the normal upgrade procedure described in

Upgrading an Enterprise HA Cluster.

2. One phase with downtime

This option requires you to add a [system property](#) to your primary node during the upgrade procedure. For details, please refer to [Upgrading an Enterprise HA Cluster](#).

If you try upgrading directly to version 5.5 **without** adding this system property, the upgrade will fail and the following message will be logged in the `artifactory.log` file:

To upgrade your HA installation to this version, you first need to upgrade to version 5.4.6 which implements changes required to accommodate a database schema change.

Highlights

Event-based Pull Replication

JFrog Artifactory now supports event based [pull replication](#), in addition to the already supported event based [push replication](#). This configuration allows your remote Artifactory instances get updated in near-real-time by a pull replication that's triggered by any changes made to your local repositories, such as new or deleted artifacts. This is great for automation purposes where you want to make your artifacts available in all of your instances as soon as they are deployed.

As a best practice, setting a [Cron expression](#) for regularly scheduled replication is still required in addition to event-based replication. This will ensure that all of the artifacts in your repository are synced and up to date, which is important in case of an event sync failure (for example, due to maintenance operations).

Native Support for SHA-256 Checksums

Artifactory now supports SHA-256 checksums. This improved algorithm to calculate checksums enables a more secure environment for your binaries letting you use SHA-256 checksums to validate the integrity of downloaded artifacts. You can also use the SHA-256 value for a variety of features as described in [SHA-256 Support](#). Whenever a new artifact is deployed, in addition to automatically calculating its MD5 and SHA1 checksums, Artifactory will now also calculate and store its SHA-256 checksum. The SHA-256 value can be used when searching for artifacts, or displayed as output for AQL queries in the same way SHA1 and MD5 checksums are used from both the UI and the REST API.

From version 5.5, Artifactory will automatically calculate the SHA-256 checksums **for new artifacts** deployed to your instance. Depending on the number of artifacts in your system, this process may take some time. To help you monitor the process, progress and status messages will be printed to a dedicated log file, `sha256_migration.log`, with some additional general messages to the `artifactory.log` file.

To maintain backward compatibility with existing scripts, the [Set Item SHA256 Checksum](#) REST API endpoint is still supported.

Feature Enhancements

Improve Performance on RPM Repositories

The performance of metadata calculation on [RPM repositories](#) has been significantly enhanced by performing different metadata calculations in parallel making resolving and deploying packages with RPM repositories much faster.

Improve Performance of NuGet Repositories

[NuGet repository](#) performance has been significantly improved when resolving dependencies or searching for artifacts. The improved performance is especially significant for repositories that host many artifacts.

Keep Multiple Versions of Metadata Files on RPM Repositories

Artifactory will now maintain previous metadata file versions on RPM repositories (primary, other, filelists) making them available for download while new ones are being generated.

This is very useful when RPM metadata is updated very frequently. If a client working with an Artifactory RPM repository downloads the `repo.md.xml` file, and the rest of the metadata files (primary, other, filelists) expire in the meantime, the expired version of these files will still be available allowing the client to complete the required download.

Retrieve Plugin Source Code by Name

Artifactory now provides access to the Groovy source code of user plugins through the [Retrieve Plugin Code](#) REST API endpoint.

Allow LDAP Users to Access Profile Page

You can now configure Artifactory to allow new users who are created by [logging in via LDAP](#) to be able to access their [profile page](#). This means that these users can now access a set of functions such as generating their API key, setting their SSH public key, configuring their JFrog Bintray credentials, and updating their password.

Support Additional MIME types in the UI

Artifactory now supports additional [MIME types](#) to allow viewing `.log`, `.yaml` and `.yml` files directly in the UI (as opposed to having to download them first). These file types are now added to the preconfigured `mimetypes.xml` file.

Enable Password Encryption by Default

For new Artifactory installations, Artifactory automatically generates a [Master Encryption Key](#) and then uses it to encrypt all passwords hosted on the instance. Decrypting passwords and encrypting them back is possible through the REST API.

To maintain consistent behavior for existing installations, upgrading to this new version will not automatically encrypt passwords.

Configurable Web Session Timeout

You can now configure Artifactory's UI session timeout using the `artifactory.ui.session.timeout.minutes` system property.

Checksum-Based Storage with S3 Object Store

Artifactory's [checksum-based storage](#) stores files in folders named after the first two characters of their checksum. When using [S3 object storage](#), this feature has been enhanced allowing you to configure the number of characters that should be used to name the folder. For example, you can configure your [S3 binary provider](#) to store objects under folders named after the first 4 characters of their checksum.

Issues Resolved

1. Fixed an issue in which Artifactory would return an error when trying to resolve an npm package because it would fail to parse an npm dependency declaration that was presented in an unexpected format.
2. Fixed an issue in which the Set Me Up screen for virtual repositories that aggregated only remote repositories would be blank.
3. Fixed an issue that caused batch download from a virtual Git LFS repository, that aggregated more than one repository, to fail.
4. Fixed an issue in which the [Build Artifacts Search](#) REST API endpoint would not return Artifacts that had been promoted to it from a different repository correctly.
5. Fixed an issue in which resolving private Docker images from a Docker remote repository that points to Docker hub failed when passwords in Artifactory were encrypted.
6. Fixed an issue in which NuGet virtual repositories that aggregated several repositories would omit results when searching for a package.
7. Fixed an issue that would sometimes cause a `NullPointerException` to be thrown when there were many deployments on a Maven repository that had a watch configured on it. The `NullPointerException` would cause metadata calculation to stop and was due to the multiple deployments causing a race condition.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.5.1

Released: September 26, 2017

This version replaces version 5.5.0 in which a critical issues was discovered.

Upgrade Notice

For an Artifactory HA installation, there are two options to upgrade to version 5.5.1 and above from a version below 5.4.6. This note only refers to upgrading Artifactory Enterprise HA installations.

Artifactory versions 5.5.1 implements a database schema change to natively support SHA-256 checksums. **If your current version is 5.4.6**, you may proceed with the normal upgrade procedure described in [Upgrading an Enterprise HA Cluster](#).

If your current version is below 5.4.6, to accommodate this change, you may select one of the following two upgrade options:

1. Two-phase, zero downtime

In this option, you first need to upgrade your HA cluster to version 5.4.6. Once this upgrade is completed, you can then proceed to upgrade your HA cluster to version 5.5.1 and above. In both phases, you follow the normal upgrade procedure described in [Upgrading an Enterprise HA Cluster](#).

2. One phase with downtime

This option requires you to add a [system property](#) to your primary node during the upgrade procedure. For details, please refer to [Upgrading an Enterprise HA Cluster](#).

If you try upgrading directly to version 5.5.1 or above **without** adding this system property, the upgrade will fail and the following message will be logged in the `artifactory.log` file:

To upgrade your HA installation to this version, you first need to upgrade to version 5.4.6 which implements changes required to accommodate a database schema change.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.5.2

Released: October 29, 2017

Highlights

Support for Acquire-By-Hash flag in Debian Repositories

Hash sum mismatch errors may sometimes cause `apt-get update` requests to Debian repositories to fail due to rotation of Debian metadata files. Artifactory now overcomes this issue by storing historical versions of the metadata files by their checksum and supporting the [Acquire-By-Hash](#) flag for Debian repositories. This allows Debian clients to download package metadata files by their checksum.

This is very useful when Debian metadata is updated very frequently. If a client working with an Artifactory Debian repository downloads the metadata files, and they expire in the meantime, the expired version of these files will still be available allowing the client to complete the required download.

Bypassing HEAD requests for remote repositories

Artifactory remote repositories normally send a HEAD request to a remote resource before downloading an artifact that should be cached. In some cases, the remote resource rejects the HEAD request even though downloading artifacts is allowed. Through the remote repository configuration, Artifactory now lets you specify that remote repositories should [skip sending HEAD requests](#) before downloading artifacts to cache.

Feature Enhancements

Automatically Rewriting External Dependencies in NPM Registries

Artifactory now supports rewriting external dependencies for various Git and GitHub URLs. For a full list of supported URLs, please refer to [Automatically Rewriting External Dependencies](#)

Issues Resolved

1. Bitbucket Server version 5.1.0 deprecated the Bitbucket Archive Plugin which remote repositories for package formats that use a Git provider in Artifactory relied on. These include Bower, VCS, CocoaPods and PHP Composer. As a result, when upgrading to Bitbucket 5.1.0, these remote repositories stopped working. This has now been fixed by adding an option to choose "Stash / Private Bitbucket (Prior to 5.1.0)" as the Git provider in the remote repository configuration for these package formats while the "Stash/Private Bitbucket" option covers Bitbucket Server version 5.1.0 and above.
2. Fixed an issue in which when executing the `/api/search/latestVersion` REST API endpoint, Artifactory would erroneously query remote repositories. This has now fixed, so Artifactory will only search in remote repositories (in addition to local and remote repository caches) when `remote = 1` is added as query param.
3. Fixed an issue in which authenticating against Artifactory Docker registries while HTTP SSO is set would fail. This has now been fixed so you can work with Artifactory Docker registries while HTTP SSO is enabled.
4. Fixed an issue in which when a REST API call included a "Range" header, the ETag returned would incorrectly include the Range provided in the header as a suffix. In turn, different clients would interpret this as a file modification. Artifactory now returns the correct ETag.
5. Fixed an issue in which system import or replication of an artifact that includes a ":" (colon) character would fail. For example, before this fix, replicating a Docker image with a LABEL that included a colon would fail.
6. Fixed an issue in which running `npm search` against an npm registry would fail if one of the packages in the results would be in the following structure: `"maintainers" : "<user name> <user email>"`, because Artifactory was expecting the structure to be: `"maintainers": [{ "name": "<user name>", "email": "<user email" }]`
7. Fixed an issue in which a 500 error would be returned when running one of the following REST API endpoints on Docker registries while using an API key for authentication:

```
/api/storage
/api/docker/{repo-key}/v2/{image name}/tags/list
/api/docker/{repo-key}/v2/_catalog
```

8. Fixed an issue which caused `checksum deploy` to sometimes fail with a 500 error. A common manifestation of this issue was replications that would fail for certain artifacts. When this error occurred, a stack trace similar to the below could be seen in the log files.

```
java.lang.NullPointerException: null
at
org.artifactory.repo.db.DbStoringRepoMixin.shouldProtectPathDeletion(DbStoringRepoMixin.java:814)
at
org.artifactory.repo.db.DbStoringRepoMixin.shouldProtectPathDeletion(DbStoringRepoMixin.java:792)
```

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.4

Released: June 20, 2017

Due to a known issue with this version, after upgrading an Artifactory HA cluster from version 5.x to 5.4.x, new nodes that you add to your Artifactory HA cluster will not start up. For a workaround, please refer to [RTFACT-14530](#).

Highlights

Access Tokens as a Separate Service

The management of [Access Tokens](#), which were introduced in Artifactory 5.0, has moved to a separate service named Access, which is installed as a separate web application. This change has no impact on how access tokens are used, however, the Artifactory installation file structure now also includes an added WAR file, `access.war`, under the `$ARTIFACTORY_HOME/webapps` folder. Artifactory communicates with the Access Service over HTTP and assumes it is running in the same Tomcat using the context path of "access".

Using access tokens through the new Access service is backwards compatible, so tokens created with earlier versions can be used for authentication with this latest version of Artifactory.

Breaking Change: Note that the change is not forwards compatible, so tokens created from version 5.4 and above cannot be used for authentication with versions previous to 5.4. This may impact a circle of trust in which some instances are running versions below 5.4 while others are running version 5.4 and above.

Running Artifactory as a service?

If you are running Artifactory as a service, once you complete the steps to upgrade to this version or later, and have replaced all files removed during the upgrade process, you need to run the `InstallService` script as described at the [end of the upgrade instructions](#).

Support for Microsoft Azure Blob Storage

JFrog Artifactory now supports [Azure Blob Storage](#) as a new object storage provider to store artifacts. Azure Blob Storage offers massively scalable enterprise storage for Artifactory supporting unstructured data of any type with strong consistency, object mutability, geo-redundancy and more. This new option opens up the opportunity to co-locate Artifactory and its storage together with all the other services that you use on the Microsoft Azure platform.

Secure Connection to Remote Repositories via SSL/TLS Client Certificates

Artifactory now supports [client certificates](#) for remote repositories facilitating secure connections with remote resources that require them (e.g., Red Hat Network that requires a Red Hat client certificate for authentication). This means that Artifactory will now be able to send the client certificate when attempting to connect to the remote resource over HTTPS.

Feature Enhancements

1. [RPM repositories](#) have been enhanced to give you control over whether the RPM file lists metadata file should be indexed by Artifactory or not. Disabling indexing of the file lists metadata improves the performance of RPM repositories with many artifacts when different clients try to resolve packages from the repository. **Note** that for new RPM repositories, indexing the file lists metadata file is disabled by default, however, when upgrading from previous versions to 5.4.0 and above, indexing for RPM repositories that already existed will remain enabled to maintain consistent behavior with the previous version.
2. Artifactory now supports the `npm login` command as a way to [authenticate the NPM client](#). Basic authentication is also still supported.
3. Previously, Artifactory was not able handle decoded slash characters in [NPM scoped packages](#), so you had to modify your reverse proxy so that it wouldn't decode the slash. Artifactory now handles decoded slash characters correctly out-of-the-box, so there is no longer any need to modify your reverse proxy.
4. Artifactory can now be configured to add [Debian packages' MD5 checksum](#) to the Packages metadata file in order to comply with the requirement of some tools (e.g. Aptly) that the MD5 is available for validation of the package.
5. The [Control Build Retention](#) REST API endpoint now accepts a query param to make deleting old builds an asynchronous process. When set, the API response acknowledges the request and outputs errors, if any, to the log.
6. The default value of the `lenientLimit` parameter for a [Sharding-Cluster Binary Provider](#) has been modified to be 1. This will allow users to continue uploading to a cluster node even if it is the only active node without having to reconfigure this parameter. Note that for filestores configured with a custom chain, the `lenientLimit` parameter will remain 0 to maintain consistency with previous versions. Therefore, the `lenientLimit` parameter will only default to 1 when using built-in templates.
7. Using the [Create Token](#) REST API endpoint, [access tokens](#) can now be created to provide the same access privileges that are given to the group of which the logged in user is a member.

Issues Resolved

1. Fixed an issue in which performing a full system import on an Artifactory HA cluster would fail. The full system import on an Artifactory HA cluster has been changed and is fully described under [System Import and Export for an HA Cluster](#).
2. Fixed an issue in which Python metadata calculation would fail if the metadata version in the METADATA or PKG-INFO files was set to 1.2.
3. Fixed an issue in which when [Enable Dependency Rewrite](#) was enabled for NPM repositories, Artifactory would only rewrite dependencies specified in the "dependencies" element of the `package.json` file and would skip the dependencies listed in the `optionalDependencies` and `devDependencies` elements.
4. Fixed an issue in which Artifactory would fail to install npm packages that contained square brackets ('[' or ']') in the "description" field of the `package.json` file.
5. Fixed an issue in which externally authenticated users (i.e. those not created in Artifactory) logging in through an external provider (e.g. LDAP) would not be able to complete artifact downloads that took a long time since the LDAP token used for authentication with Artifactory would expire. This was fixed by exposing the `artifactory.artifactory.tokens.cache.idleTimeSecs` system property that managed this timeout and increasing its default value from 5 minutes to 10 minutes.
6. Fixed an issue in which existing repositories enabled for indexing by JFrog Xray did not trigger indexing automatically and required you to manually trigger indexing through the JFrog Xray UI or using the REST API.
7. Fixed an issue in which using `mvn site-deploy` with the `maven-site-plugin` to upload a site to Artifactory would fail when the site's URL contained a dot (".") in its path (e.g. `libs-snapshot-local./file.jar`)
8. Fixed an issue in which NuGet virtual repositories that aggregated more than one local or remote repository would omit results or return duplicate results when searching for a package.
9. Fixed an issue in which Artifactory 5.x would not display certain builds in the UI because it failed to parse dates presented in [ISO 8601](#) format (e.g. 2016-09-08T21:02:17.781+03:00)
10. Fixed an issue in which upload to a repository would fail, if an event-based replication defined for the repository failed for any reason. Following the fix, uploading a file to the repository succeeds even if replication fails.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.4.1

Released: June 22, 2017

Issues Resolved

1. Fixed an issue in which the schema version of a Docker image manifest would change from 2 to 1 when the image was distributed from Artifactory to JFrog Bintray.
2. Fixed an issue that caused batch downloads from a virtual Git LFS repository that aggregated both local and remote repositories to fail.

This happened when Artifactory would find one of the files in an aggregated local repository (and therefore should have stopped searching for it), but would still go on to search for it in the aggregated remote repositories. If the file did not exist in any of the remote repositories, Artifactory would not serve the file.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.4.2

Released: June 30, 2017

Issues Resolved

1. Fixed an issue in which Artifactory failed to start up when Tomcat was configured to only serve HTTPS content, or was configured to serve both HTTP and HTTPS, but on different ports.
2. Fixed an issue in which when an Artifactory HA installation's filestore configuration used the eventual-cluster binary provider (for example, when using one of the cloud storage providers), in rare cases, when uploading files involving a large number of transactions, Artifactory would indicate that files were successfully uploaded to storage, when in fact, the uploads failed.
3. Fixed an issue in which Artifactory was unable to connect to the Access Service (and as a result failed to start) when Tomcat was configured with a self signed chain certificate.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.4.3

Released: July 3, 2017

Due to a known issue with this version, after upgrading an Artifactory HA cluster from version 5.x to 5.4.x, new nodes that you add to your Artifactory HA cluster will not start up. For a workaround, please refer to [RTFACT-14530](#).

Issues Resolved

1. Fixed an issue in which uploading or downloading files to Artifactory using access tokens may have failed with error 500. This happened when running Artifactory 5.4.2 and using access tokens with a **subject** that was longer than 64 characters.
2. Fixed an issue in which upgrading an RPM or Debian installation of Artifactory that use the `systemd` init system would fail with a "The currently installed Artifactory version does not have the same layout as this DEB!" error.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.4.4

Released: July 6, 2017

Issues Resolved

1. Fixed an issue in which after upgrading an Artifactory HA cluster from version 5.x to 5.4.x, new nodes that were added to the Artifactory HA cluster would not start up.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.4.5

Released: July 18, 2017

Highlights

Puppet Repositories Support librarian-puppet and r10k

Artifactory's support for [Puppet repositories](#) has been significantly upgraded by introducing support for `librarian-puppet` and `r10k` allowing extended configuration management with these popular Puppet clients. In addition, Artifactory also exposes new REST API

endpoints to retrieve Puppet modules and releases to facilitate automated configuration management using Puppet.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.4.6

Released: August 7, 2017

Feature Enhancements

Support Pagination for Docker v2 APIs

Artifactory now supports pagination when [listing Docker image tags](#) and retrieving a registry's [catalog](#) using the REST API. This can be useful for automation purposes and Docker clients that use pagination parameters.

Issues Resolved

1. Fixed an issue in which when resolving a package from an npm repository, Artifactory would throw a deserialize error to the log file if one of the package's dependencies in the corresponding `package.json` file was declared using the following format: "`<dependency_name> : { "version" : "<version_number>" }`".
For example: the "deep-diff" package uses this format. As a result, the npm client would fail to resolve the package.
2. Fixed an issue that prevented using Git LFS client v1.x with [Git LFS repositories](#) in Artifactory when using SSH.
3. Fixed an issue in which NuGet virtual repositories that aggregated several repositories would omit search results when searching for a package.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.3

Released: May 11, 2017

Due to a critical issue, if you are upgrading from a version below 4.4.1 directly to version 5.3, Artifactory will fail to start up. A patch has been released, and if your current version is below 4.4.1 you should upgrade to [Artifactory 5.3.1](#).

Highlights

Grant Admin Privileges to a Group of Users

Artifactory now supports granting Admin privileges to a group of users which greatly improves the user experience since previously you could only provide Admin privileges to users individually.

This allows you to import a group from your LDAP or Crowd server and [grant Admin privileges](#) to the whole group in a single action.

Automatically Associate a SAML SSO User to an Artifactory Group

Artifactory will now accept a custom SAML attribute that can be mapped to existing groups (including imported LDAP groups). If a SAML user has the custom SAML attribute he will now inherit the permission specified in the corresponding group in Artifactory for the current login session.

Feature Enhancements

1. Performance of displaying data in the **Builds** module in Artifactory UI has been significantly improved. This creates a much better user experience, especially for Artifactory instances with many builds or when viewing a project with many builds.
2. When importing users via SAML SSO, the users' email addresses are now also fetched and populate the corresponding field in their Artifactory user profile.
3. The installation script that installs Artifactory as a service has been enhanced to use `systemd` on Linux distributions that support it. The script will automatically detect if `systemd` is supported, and if not, will use `init.d` as currently implemented.
4. In the Tree Browser, when selecting the **Effective Permissions** tab for the selected repository, you may now view the permission targets associated with that repository.
5. Previously, virtual repositories would only provide a **General** tab with basic information about selected artifacts. Now, virtual

repositories provide additional tabs that show all data about artifacts selected similar to the data that is provided when selecting the artifacts directly from the aggregated local or remote repositories.

Issues Resolved

1. Fixed an issue that prevented using Git LFS client v2.x with **Git LFS repositories** in Artifactory when using SSH.
2. Fixed a resource leak that was introduced when "Enable Dependency Rewrite" was enabled in **virtual NPM repositories**. This issue may have caused depletion of different resources such as open file handles, database connections and storage streams.
3. Fixed an issue that prevented pushing or pulling Docker images that had foreign layers when the image also had a "history" field in its *config.json* file.
4. Fixed an issue that caused a login failure when the "List Contents" permission in Active Directory was enabled for an Admin, but not for the user that was attempting to log in.
5. Fixed an issue related to Maven repositories in which the wrong artifact may have been retrieved for a download request since Artifactory did not consider the full path beyond the GAV coordinates.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.3.1

Released: May 24, 2017

Highlights

This is a patch that fixes a critical issue that was discovered in version 5.3.0 in which after upgrading from a version **below Artifactory 4.4.1 directly to Artifactory 5.3.0**, Artifactory failed to start up.

Note that this issue did **not** affect upgrades from Artifactory 4.4.1 and above.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.3.2

Released: June 7, 2017

Issues Resolved

1. Fixed an issue in which, when upgrading an Artifactory HA cluster with 2 or more nodes, from version 5.x to version 5.3.x, Artifactory would throw a HazelcastSerializationException when displaying the UI. In the process of upgrading the cluster, you will still encounter this issue from nodes that have not yet been upgraded.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.2

Released: March 28, 2017

Main Updates

1. Improved the performance of property search when using PostgreSQL.
This will significantly improve Docker operations on Artifactory Docker registries as the property search mechanism is used upon searching for Docker layers.
2. Improved the performance of Docker layers search mechanism on Artifactory Docker registries. This will be mostly significant when working with Docker layers that are being used by thousands of Docker images.
3. The Tomcat bundled with Artifactory has been upgraded to version 8.0.41.
4. Artifactory now regards the *content.xml.xz* and the *artifacts.xml.xz* files on a remote P2 repository as expirable resources, so whenever there is a metadata change in one of these files, Artifactory will use the updated file instead of the expired one.
5. When working with Conan repositories, Artifactory now supports variables with multiple values in the *conanfile.txt* file. This enables Artifactory to fully extract **[env]** variables with multiple values and assign all those values to the corresponding property annotating the package in Artifactory.
6. Fixed an issue in which deploying multiple files to a virtual repository through the UI would fail.
7. Fixed a bug related to remote Docker registries in Artifactory that left connections and input streams open following docker pull

operations.

8. Fixed an issue related to Debian repositories. Artifactory now adds an empty line at the end of the **Packages** file to fully support Debian tools such as `debootstrap`.
9. Fixed an issue related to Debian repositories in which the **Components** section in the generated **Release** file was named "Component" when there was indeed only one component. This has been fixed by naming the section "Components", regardless of the number of components. Following the fix, Artifactory now fully support tools such as `debootstrap`.
10. Fixed an issue occurring in Artifactory HA clusters. When a node was stopped for any reason, its state as reported by the UI remained as **Running**. This has now been fixed so the state for a stopped node is displayed as **Unavailable**.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.2.1

Released: April 13, 2017

Highlights

Access Tokens

Authentication using [access tokens](#) has undergone two significant enhancements.

1. Any valid user in Artifactory can now create access tokens for personal use whereas previously only an Artifactory admin could create access tokens. This removes the burden of creating and managing access tokens for all users from the admin's shoulder, and gives non-admin users more freedom to operate within their ecosystem.
2. An Artifactory administrator can now create access tokens with admin privileges whereas previously, access privileges were specified by inclusion in different groups. This enhances the integration of external applications which may need admin privileges to work seamlessly with Artifactory.

Feature Enhancements

1. When upgrading an Artifactory HA installation from version 4.x to version 5.x, managing the [bootstrap bundle](#) has been improved to become an automatic and seamless process. Artifactory will now create the [bootstrap bundle](#) on the primary node automatically, and extract it to the secondary nodes, so there is no longer any need to create and copy the bootstrap bundle manually.
2. [Control Build Retention](#) : A new REST endpoint that lets you specify parameters for build retention has been added. Previously build retention could only be specified when uploading new build info. This enhancement provides an easy way to configure cleanup procedures for different jobs, and reduces the risk of timing out when deploying heavy build info.
3. By default, the "latest" version of an NPM package is the one with the highest SemVer version number. NPM repositories have now been enhanced so you can override the default behavior by setting a system property to assign a "latest" tag to the package that was most recently uploaded.
4. The [Artifactory Docker image](#) now comes with the PostgreSQL driver built in, so there is no need to mount it separately or build it into a separate Docker image.

Issues Resolved

1. Artifactory is now aligned with the Docker spec and returns an authentication challenge for each Docker endpoint (even when anonymous access is enabled). This means that when using internal Artifactory Docker endpoints, you must first retrieve an authentication token which must then be used for all subsequent calls by your Docker client.
2. Fixed an issue in which NuGet virtual repositories that aggregated more than one local or remote repository may have omitted results when searching for a package.
3. When an Artifactory user with no "Delete" permissions was trying to deploy a build while specifying build retention, Artifactory would try and delete old builds and return a 500 error. This has now been fixed, and Artifactory will, instead, return a 403 error.
4. Fixed an issue in which Artifactory failed to pull a Docker image according to the digest of the manifest file from a remote Docker registry.
5. Fixed an issue in which aborting a download of a folder as an archive could leave open connections that were not closed which in turn would prevent further download of folders.
This has now been fixed so download slots are freed and the connection is closed properly.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.1

Released: February 21, 2017

Configuration Management with Chef

Artifactory meets the heart of DevOps adding full support for configuration management with Chef. Share and distribute proprietary Cookbooks in local Chef Cookbook repositories, and proxy remote Chef supermarkets and cache remote cookbooks locally with remote repositories. Virtual Cookbook repositories let you access multiple Cookbook repositories through a single URL overcoming the limitation of the Knife client that can only access one repository at a time.

Configuration Management with Puppet

Artifactory now also fully supports configuration management with Puppet. Use local Puppet repositories to share and distribute proprietary Puppet modules, and use remote Puppet repositories to proxy and cache Puppet Forge and other remote Puppet resources. Use a virtual Puppet repository so the Puppet client can access multiple repositories from a single URL.

Main Updates

1. Support configuration management with Chef through [Chef Cookbook repositories](#). Artifactory fully supports the Knife client for authenticated access, and also supports Berkshelf for anonymous access. Authenticated access for Berkshelf will be added in a forthcoming release.
2. Support configuration management with Puppet through [Puppet repositories](#). Full support for Puppet command line along with local, remote and virtual repositories for hosting and provisioning Puppet modules.
3. For Artifactory administrators, a list of common actions is available from the [top ribbon](#) in the Artifactory UI for quick and easy access. This makes it easy to do things like creating repositories, adding users, adding groups and more.
4. Artifactory can now be run as a standalone instance in a Kubernetes cluster. For details, please refer to [JFrog's examples using Docker on GitHub](#).
5. Artifactory now supports disabling UI access (i.e. the user may only access Artifactory through the REST API) through the addition of the `disableUIAccess` element in the [Security Configuration JSON](#).
6. The default order of repository types in the tree browser has been changed to show virtual and distribution repositories first, as these are accessed more frequently, and then local and remote repositories.
7. Modified [NGINX reverse proxy configuration generated by Artifactory](#) to enable using NPM scoped packages.
8. A performance issue with the login and logout procedure has been fixed, so the time to login or logout is now significantly reduced.
9. A bug in which duplicate files simultaneously uploaded to a sharded filestore occasionally caused deletion of the files, was fixed.
10. A bug in permissions management that disabled the Admin module after removing the default "Anything" and "Anonymous" permissions, was fixed.
11. Fixed an issue when upgrading Artifactory 4.x to 5.x in which the IAM role settings for S3 object storage in the `binarystore.xml` were not correctly migrated to the upgrade has been fixed.

For a complete list of changes please refer to our [JIRA Release Notes](#)

Artifactory 5.1.2

Released: March 8, 2017

Note: Due to a critical issue found when uploading files larger than 100MB to S3 compatible storage, this version has been removed from [JFrog Bintray](#).

Main Updates

1. Fixed a performance issue related to the "Most Downloaded Artifacts" widget on the Artifactory Home Page which, when refreshed, could cause the Artifactory database to stall on instances with a large number of artifacts.
2. Added support for Conan client v0.20.0 which includes a new section in the conanfile to allow adding environment variables and custom properties. These are indexed in Artifactory as properties and can be used in searches.
3. Improved performance of queries for artifacts which include an underscore character ("_") in their name. This is especially important for resolution of Docker images since all Docker layers include an underscore in the layer name.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.1.3

Released: March 9, 2017

Main Updates

1. Fixed issue related to uploading files larger than 100MB to S3 bucket.
2. Fixed issue causing display wrong information in "Most Downloaded Artifacts" when working with OracleDB.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.1.4

Released: March 19, 2017

Main Updates

1. Fixed an issue preventing Artifactory from starting up following an upgrade to version 5.x on Windows when Artifactory is configured with a [Keystore](#).

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.0

Released: January 31, 2017

Improvements in Artifactory HA

- **Cloud Native Storage:** Artifactory HA infrastructure has undergone significant changes and now fully supports cloud native storage. We have completely removed the requirement for using a Network File System (NFS). This release introduces a new type of binary provider that manages distribution of files and configuration across the cluster nodes. This new functionality supports scaling out your storage by relying on object storage solutions or using the nodes' filesystem without the limitations of a traditional NFS, while enjoying other benefits such as distributed storage and redundancy.
- **Removal of Sticky Sessions:** Artifactory no longer requires that the load balancer used in the Artifactory HA network configuration support session affinity (sticky sessions). You may need to change or remove NGINX configurations that related to sticky sessions.
- **Cluster License Management:** Managing licenses for an Artifactory HA cluster is much simpler in Artifactory 5.x. Instead of registering a license per node, just upload all your cluster license keys to any cluster node, and Artifactory will transparently allocate them as new nodes are added to and removed from the cluster. This allows automatic provisioning of cluster nodes without the need to deal with manually assigning a license for each node.

Compatibility with JFrog Mission Control

If you are managing your Artifactory licenses through JFrog Mission Control, Cluster License Management will also be supported in Mission Control, starting from version 1.8, scheduled for release with the next release of Artifactory which is scheduled for February 2017.

To perform a clean installation of Artifactory HA, please refer to [HA Installation and Setup](#).

To upgrade your current installation of Artifactory HA, please refer to [Upgrading Artifactory HA](#).

Running Artifactory as a Docker Container

Installing and running the [Artifactory Docker image](#) has been greatly simplified. Essentially it is now a matter of running `docker pull` and then `docker run`, while passing in mounted volumes to maintain persistence.

Access Tokens

Artifactory 5.0 introduces [access tokens](#) as a new and flexible means of authentication allowing cross-instance authentication, authenticating both users and non-users, providing time-based access control and group-level access control.

Enriched and Simplified Onboarding Experience

When starting up for the first time, Artifactory presents two new ways to get you through basic setup and configuration so you can get started immediately. The first is the [Onboarding Wizard](#) that creates default repositories for package types you select, sets up a reverse proxy, sets the Admin password and more. The second is a [YAML Configuration File](#) in which you can configure the same parameters that the wizard is used for. For example, once you have configured your first instance of Artifactory through the Onboarding Wizard, you can generate the YAML Configuration File from it and use that to spin up additional instances with the same initial configuration.

New Home Screen

The Artifactory [Home Screen](#) has been completely redesigned in version 5.0. The new Home Screen provides quick and easy access to some of the most common actions taken by users including searching for artifacts using any of the search methods available, creating new repositories, displaying the "Set Me Up" dialog for any repository, showing information on the latest builds and downloaded artifacts and more.

Breaking Changes

Artifactory HA Infrastructure has Undergone Several Changes

- **Removal of NFS requirement:** Previously, Artifactory HA required setting up a mount that was used by the `$CLUSTER_HOME` folder to synchronize configuration and binary files between the cluster nodes. This requirement is now removed. Configuration files are maintained in the database, and binaries may be stored on alternative storage such as local storage on each node or on a cloud storage provider. To learn how to migrate your filestore from NFS to alternative storage, please refer to [Migrating Data from NFS](#).
- **Bootstrap Bundle:** When setting up an HA cluster, you need to create a [bootstrap bundle](#) on the primary node, and then copy it to each secondary node you add to the cluster before starting it up.
- **License Management:** Artifactory HA licenses are now fully managed through the [Cluster License Manager](#).
- **Unlicensed Nodes:** When adding and starting up a node, if a valid license is not available to the Cluster License Manager, the node will continue to run, but will remain unlicensed and return a 503 error to any requests it receives. To keep your HA cluster running until the node is licensed, you can modify your reverse proxy configuration to redirect requests to the next upstream if a 503 error is received by adding

```
proxy_next_upstream http_503 non_idempotent;
```

Please refer to [Configuring a Reverse Proxy](#) where you can [generate](#) a new Reverse Proxy Configuration that includes the modification needed.

Black Duck Code Center Integration Deprecated

Artifactory's direct integration with Black Duck Code Center has been deprecated. To continue using the Black Duck service, you can connect Artifactory to JFrog Xray which has [integrated with Black Duck](#) as an external provider of issues and vulnerabilities.

Global /repo Repository Deprecated

The Artifactory `/repo` repository endpoint is being deprecated. As part of the deprecation, any requests to the global `/repo` repository will no longer be valid, regardless to the value of the `artifactory.repo.global.disabled` system property. If you believe this deprecation will affect existing build jobs or scripts that are referencing the global repo, due to the deprecation, you will now be able to create your own standard Virtual Repository and call it "repo", since the name will no longer be reserved.

Change in Startup and Shutdown Scripts

The startup and shutdown scripts have changed in Artifactory 5.0. Previously, these scripts used to create the "Artifactory" user as a standard user. To improve security, the user is now created without a login shell and the execution scripts use "su -s" (instead of "su -") which means that the Artifactory user will not be available for any purpose other than for startup and shutdown.

Set Item Properties REST API Endpoint Changed

The version of Tomcat used in Artifactory 5.0 has been upgraded to **8.0.39**. This version of Tomcat no longer supports unencoded URLs, so the REST API endpoints which used a pipe character ("|") as a separator have undergone corresponding changes so you can use a semicolon (";") as a separator instead, or use escaping to represent a pipe as `%7C`. Any scripts that use these endpoints may have to be changed to support the new specification. For details, please refer to [Set Item Properties](#) as an example.

Session ID Cookie Changed

Your Artifactory session ID is now stored in a `SESSION` cookie (instead of a `JSESSIONID` cookie).

Main Updates

1. Artifactory can now be installed in a [High Availability configuration](#) without needing an NFS.
2. [Cluster License Manager](#) for Artifactory HA installations automatically manages licensing for your cluster nodes. This will also be supported by JFrog Mission Control in its forthcoming release.
3. Greatly simplified Artifactory [Docker image installation](#).
4. Authentication using [Access Tokens](#).
5. [Greatly simplified onboarding](#) using either a UI wizard or a YAML file.
6. [Home Screen](#) has been redesigned with a new look and feel for easy access to common actions and a rich user experience.
7. [Search](#) has been redesigned and is now available as a separate module for easy access from anywhere.
8. UI notifications in Artifactory have been improved for clarity.
9. [Monitoring Storage](#) is updated with a new look and feel.
10. Removed the requirement for session affinity in the load balancer used in an Artifactory HA cluster.
11. Direct integration with Black Duck has been deprecated. You may continue using [Black Duck through JFrog Xray](#).
12. Global `/repo` repository has been deprecated.
13. Artifactory Tomcat version was upgraded to 8.0.39.
14. From version 5, the YUM package type is replaced with RPM. i.e. what used to be a YUM repository is now referred to as an [RPM repository](#). YUM will continue to be supported as a package type when creating repositories through the REST API for backward compatibility.
15. Users who are logged in through a [SAML](#) server can be associated with [LDAP groups](#) through the use of a user plugin. Use [this user](#)

[plugin](#) as a reference as an example of a user plugin.

16. LDAP login performance was improved by narrowing Arifactory's search filter so it only searches through groups that have been imported to Artifactory rather than the full set of LDAP groups.
17. Added support for [Docker manifest](#) to reference remote layers by URL that will be pulled by the Docker engine before running the image.
18. Added metadata validation for Debian packages to ensure [Debian repositories](#) are not corrupted by malformed packages.
19. Fixed an issue in which [Docker images](#) which were imported to Artifactory and then exported sometimes failed to produce the correct schema.
20. Fixed an issue regarding email notifications for backups so that now, a notification is sent for both manual and automatic scheduled backups if the backup fails.
21. Fixed an issue in which downloading from a virtual [Git LFS repository](#) would fail if the file would not exist in the first positioned repository in the list.
22. Fixed an issue in which YUM metadata GPG signing was skipped if the passwords in Artifactory were encrypted.
23. Fixed an issue in which [Git LFS repositories](#) that require authentication will fail push requests when Anonymous Access is enabled.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 5.0.1

Released: February 7, 2017

Main Updates

1. A memory leak that was discovered in the new [cluster license manager](#) implementation has been fixed. This issue may have caused Artifactory to stop responding and is now resolved.
2. A limitation in Artifactory HA, that potentially prevented you from accessing large support bundles, and prevented Artifactory from starting up, has been removed. Now, you can access the support bundle for any node in an HA cluster regardless of its size.
3. An issue preventing Artifactory from starting up when using IBM JDK 8 has been fixed.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.16

Released: January 16, 2017

Support for Xray CI/CD Integration

As a critical link between JFrog Xray and Jenkins CI (more CI servers will be added in future releases), Artifactory adds support for Xray's CI/CD integration allowing you to fail build jobs if vulnerabilities are found in the build. Artifactory acts as an intermediary between Jenkins and JFrog Xray.

You can configure the Jenkins Pipeline to send a synchronous request to Xray to scan a build that has been uploaded to Artifactory. Artifactory passes the request on to Xray which scans the builds in accordance with Watches you defined, and respond with an indication to fail the build job if an alert is triggered.

Xray CI/CD integration is supported from Artifactory 4.16, JFrog Xray 1.6 and Jenkins Artifactory Plugin 2.9.0.

Main Updates

1. Add support for JFrog Xray CI/CD integration allowing you to fail build jobs if the build scan triggered an alert.
2. Fix a bug that caused a memory leak related to JFrog Mission Control DR configuration.
3. Fix an issue in which `createdBy` and `modifiedBy` fields were missing after running an import.
4. When a build is deleted, whether through the UI, via REST API or due to a build retention policy, Artifactory now sends a corresponding event to Xray so it can remove that build from its database and avoid triggering alerts for deleted builds.
5. A fix has been put in place to prevent a security vulnerability (CVE-2016-10036) that may have been exploited through a web UI API endpoint, which potentially allowed unauthorized uploading of files to unexposed locations in the Artifactory host. JFrog would like to thank **Alessio Sergi** of Verizon Enterprise Solutions for reporting this issue and for working with JFrog to help protect our customers.

Artifactory 4.16.1

Released: March 15, 2017

Main Updates

1. The Tomcat bundled with Artifactory has been upgraded to version 8.0.41.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.15

Released: December 13, 2016

Conan Repositories

Artifactory brings binary repositories to the world of C/C++ development with support for [Conan repositories](#). By supporting the Conan client, Artifactory offers enterprise grade repository management supporting high-availability, fine-grained access control, multi-site development, CI integration and more. Providing an in-house local repository for C/C++ binaries, Artifactory is a secure, robust source of dependencies and a target to efficiently upload packages built through Conan. C/C++ development will never be the same again.

Main Updates

1. Add support for [Conan repositories](#).
2. Significantly improved performance in Artifactory installations serving thousands of users related to the intensive permission validation process. For example, this should solve slow NuGet search issues in these Artifactory installations.
3. Fixed an issue in which changing the severity specified for [download blocking](#) for a repository, or removing it altogether, did not update Xray correctly and the change was not registered.
4. Fixed an issue in which the JSON returned from [Get Repository Replication Configuration](#) was not always compatible with REST API endpoints used to set repository replication configuration.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.14

Released: October 20, 2016

PHP Composer Repositories

Artifactory now supports development with PHP as a fully-fledged PHP Composer repository. Create local repositories to host your internal PHP packages, or proxy remote resources that host PHP index files or PHP binary packages.

Main Updates

1. Support [PHP Composer](#) local and remote repositories.
2. Artifactory can now issue a warning before running a backup if there is [insufficient disk space](#).
3. Performance when simultaneously calculating Debian metadata for multiple distributions in multiple repositories has been improved.

Known Issues

1. In case DR instance is managed by JFrog Mission Control there is a risk of a memory leak which may cause the Artifactory service to stop responding.
Related issues are [RTFACT-12854](#), [RTFACT-13358](#).

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.14.1

Released: November 1, 2016

Main Updates

1. Fixed an issue related to clean up of YUM metadata index files.
2. Fixed a distribution issue related to packages with special characters (e.g. '!') in the package or version name.

Known Issues

1. In case DR instance is managed by JFrog Mission Control there is a risk of a memory leak which may cause the Artifactory service to stop responding.

Related issues are [RTFACT-12854](#), [RTFACT-13358](#).

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.14.2

Released: November 27, 2016

Main Updates

1. **LDAP login performance improved**

Login performance has now been improved by only searching attributes that have been configured in the LDAP Group setting rather than for the entire set of attributes. This is especially noticeable when user belongs to many groups.

2. **Npm search issue fix**

Due to breaking changes in npm client behavior, from version 4.0 of the Npm client, searching through Artifactory was failing. This was because the client could not parse the response with the "_updated" field of searches that used "since" . This has now been fixed by removing the field from the response for partial searches.

3. **NuGet search issue fix**

When the results of NuGet package search required pagination, several results were omitted. This was due to a mismatch between how Artifactory returned each page of the results (using a "\$skip" parameter), and how the NuGet client expected the result (based on the "\$top" parameter. This has now been fixed by aligning Artifactory with the NuGet client so no results are omitted.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.14.3

Released: December 7, 2016

Using Previous Encryption Keys

If Artifactory is unable to decrypt data with the current Master Key (the contents of the `artifactory.key` file), you can now set the `artifactory.security.master.key.numOfFallbackKeys` property in the `artifactory.system.properties` file which specifies the number of previous keys Artifactory should try and use to decrypt data .

Main Updates

1. Enable Artifactory to use previous Master Keys keys to decrypt data.

Known Issues

1. In case DR instance is manage by JFrog Mission Control there is a risk of a memory leak which may cause the Artifactory service to stop responding.
Related issues are [RTFACT-12854](#), [RTFACT-13358](#).

For a complete list of changes please refer to our [JIRA Release Notes](#) .

Artifactory 4.13

Released: September 21, 2016

Xray Enhancements

- **Global enable/disable:** Globally enable or disable the Xray integration
- **Download blocking:** When connected to JFrog Xray, Artifactory can be configured per repository to block download of artifacts that have not yet been scanned, or those that have been scanned and identified to include issues of a given severity
- **Scan specific artifact or path:** Initiate scanning and indexing of a specific artifact or path selected in the tree browser

JMX MBeans to support monitoring of log appenders for log analytics

Artifactory now implements MBeans that let you monitor appenders that send [log information](#) to Sumo Logic for log analytics.

Main Updates

1. Enhancements to the Xray integration including globally enabling or disabling the integration, download blocking and specific artifact/path scanning.
2. JMX MBeans that monitor appenders that send [log data](#) to Sumo Logic for log analytics.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.13.1

Released: October 13, 2016

Main Updates

1. An issue, in which Bower packages downloaded from virtual repositories were returned "flat" rather than in their original structure, has been fixed.
2. The [system logs](#) are refreshed periodically. An administrator can now pause the countdown to refresh the system log.
3. The order in which different repository types are [sorted in the tree browser](#) can now be set by a system property.
4. Performance when managing Groups and Users for permission targets has been improved.

Known Issues

1. In case DR instance is managed by JFrog Mission Control there is a risk of a memory leak which may cause the Artifactory service to stop responding.
Related issues are [RTFACT-12854](#), [RTFACT-13358](#).

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.13.2

Released: October 18, 2016

Main Updates

1. Fixed security issue and minor bugs.

Known Issues

1. In case DR instance is managed by JFrog Mission Control there is a risk of a memory leak which may cause the Artifactory service to stop responding.
Related issues are [RTFACT-12854](#), [RTFACT-13358](#).

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.12.0.1

Released: August 29, 2016

Note: This release replaces version 4.12.0 due to a critical issue that was found.

JMX MBeans

To monitor resource usage, Artifactory now implements JMX MBeans that monitor HTTP connections. This exposes a variety of new parameters that you can monitor such as remote repositories, JFrog Xray client connection, distribution repositories, replication queries, HA event propagation and more.

YUM Virtual Repositories

With support for virtual YUM repositories, you can both download and upload RPMs using a single URL.

Main Updates

1. Support [YUM Virtual Repositories](#).
2. JMX MBeans support has been expanded to allow monitoring HTTP connections.
3. A remote repository and its corresponding cache are now collated in the [Artifact Repository Browser](#) and displayed together rather than in separate sections.
4. As a convenience feature, you can now filter users to be removed from a group or repositories to be removed from a permission target.
5. Hazelcast interface matching has been disabled, allowing you to run Artifactory HA cluster nodes under different Docker hosts.
6. A `targetInfo` variable has been added to the [Replication User Plugin](#) context allowing you to specify the target Artifactory URL and repository.
7. Performance of RubyGems `api/dependencies` queries has been improved.
8. Push replication now supports synchronizing download stats (for local repositories). To avoid inadvertent deletion artifacts, this is recommended when setting up replication for disaster recovery.

Known Issues

1. When pushing existing docker layers to using to deploy to virtual layers will be uploaded to the wrong path. The path will be prefixed with the target local repository key.
Note that pull command will continue to work as expected.
Related issue is [RTFACT-12396](#), fixed in version 4.12.1.
2. RubyGems dependency query might cause unexpected DB behavior when working with a very large sets of artifacts.
Related issue is [RTFACT-12480](#), fixed in version 4.12.2.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.12.1

Released: September 7, 2016

Main Updates

1. Fix an issue that caused existing Docker layers to be uploaded to the wrong path when deploying to a virtual repository.
This patch will also include a conversion to move layers from the wrong path to the correct path.
2. Fix "AWS EC2 IAM SessionCredentials" refresh token process, when using IAM role and time is set to any time zone other than GMT.

Known Issues

1. RubyGems dependency query might cause unexpected DB behavior when working with a very large sets of artifacts.
Related issue is [RTFACT-12480](#), fixed in version 4.12.2.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.12.2

Released: September 14, 2016

Main Updates

1. Fix an issue causing DB to behave unexpectedly when using `/api/gem/dependencies` query on RubyGems repositories with a very large set of artifacts.
2. Fix an internal server error on "[Artifacts Not Downloaded Since](#)" REST api.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.11

Released: July 31, 2016

JFrog Xray Integration

The first official version of JFrog Xray, version 1.0 has been co-released with this version of Artifactory. JFrog Xray 1.0 supports Artifactory 4.11, and above.

To integrate JFrog Artifactory 4.11 with JFrog Xray 1.0 you need to take the following steps:

- If you are doing a clean installation of JFrog Artifactory 4.11, follow the usual instructions under [Installing Artifactory](#), and then install JFrog Xray as described in the [JFrog Xray User Guide](#).
- If you are upgrading from a previous version of JFrog Artifactory **to which you had connected the JFrog Xray preview version**, please follow [these instructions](#) to create a clean environment for installation.

Performance

This version presents several improvements in performance including deletion of an artifact's properties, garbage collection and data import and restoring artifacts from the trash can.

Main Updates

1. Performance when making many changes (e.g. Delete all) to an artifact's properties has been greatly improved.
2. Performance of the trash can has been greatly improved both when deleting artifacts or restoring them from the trash can.
3. Garbage collection and data import performance has been greatly improved by separating these two actions in different threads.
4. For artifacts that are indexed by JFrog Xray, the **General** tab in the tree browser now displays Xray indexing and status information.
5. [Repository Configuration](#) REST API endpoint has been updated to provide caller with the same information that is available, according to that user's permissions, when querying a repository through the UI .
6. A fix has been put in place to prevent a security issue due to "LDAP Attribute Poisoning" (CVE-2016-6501). JFrog would like to thank **Alvaro Munoz** and **Oleksandr Mirosh** of Hewlett Packard Enterprise for reporting this issue and for working with JFrog to help protect our customers.

Known Issues

1. Null pointer exception error is thrown when a property has a NULL value (RTFACT-12058). This might be caused by YUM metadata calculation when a YUM group is being used causing the vendor value to be NULL. As a workaround for this issue you can set the following system property **artifactory.node.properties.replace.all=true** under \$ARTIFACTORY_HOME/etc/artifactory.system.properties and restart Artifactory service. (in case you are using High Availability set up this change need to be done on each node).

Make sure to change the value back to false after you upgrade to a later version since this issue is already fixed and leaving it to true will result in Artifactory not using the new properties update mechanism.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.11.1

Released: August 14, 2016

Improvements to Docker Registries

Several improvements have been made for Docker registries in Artifactory.

- Pull replication for remote Docker repositories, that was previously not possible due to a limitation in the Docker client, has now been enabled for images created with the manifest schema v2.
- Storage of Docker images has been optimized so that Artifactory will not duplicate layers of a Docker image that is pushed if those layers already exist elsewhere in Artifactory.

Main Updates

1. In addition to listing files in Amazon S3 storage, Artifactory can now also list files in Google S3 storage.
2. Pull replication has now been enabled for Docker registries for images created with manifest schema v2.
3. When pushing a Docker image that contains layers that already exist, Artifactory will use the existing layers rather than storing an additional copy.
4. Artifactory now supports [GPG signing](#) for YUM metadata
5. AQL can now be invoked from user plugins related to search.
6. Artifactory is now available for installation as a Debian distribution for Xenial (Ubuntu 16.04).

Known Issues

1. When pushing existing docker layers to using to deploy to virtual layers will be uploaded to the wrong path. The path will be prefixed with the target local repository key.
Note that pull command will continue to work as expected.
Related issue is [RTFACT-12396](#), fixed in version 4.12.1.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.11.2

Released: August 17, 2016

Main Updates

1. Fix sending unnecessary delete event to Xray when overriding file with the same checksum.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.10

Released: July 19, 2016

Log Analytics

This version introduces the capability for integration with Sumo Logic Log Analytics. Artifactory creates an account with Sumo Logic so you can view advanced analytics of your Artifactory logs to discover performance bottlenecks, attempts at unauthorized server access and more.

Docker Image Cleanup

You can now configure how many snapshots of each docker image tag Artifactory should store before deleting old snapshots to avoid them accumulating and bloating your filestore.

Main Updates

1. Integration with Sumo Logic for [Log Analytics](#).
2. Configure Artifactory to automatically cleanup old tags of Docker images by limiting the number of unique tags stored in any Docker registry in Artifactory.
3. Performance of Maven metadata calculation has been improved to accommodate many delete operations on a Maven repository.
4. A new navigation menu with major improvements in the **Admin** module allowing you quickly filter and navigate to a specific category. The **full menu** is displayed on a mouse-over, and you can enter a search term to emphasize the item you are looking for.
5. Support retagging a Docker image as part of the Docker promotion REST API, enabling you to easily rename and retag an image without having to pull and push it again. This is very useful when using promotion to manage your CI pipeline.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.9

Released: July 3, 2016

JFrog Xray Integration

This version introduces the capability for full integration with JFrog Xray, Universal Artifact Analysis, that reveals a variety of issues at any stage of the software application lifecycle. By scanning binary artifacts and their metadata, recursively going through dependencies at any level, JFrog Xray provides radical transparency and unprecedented insight into issues that may be lurking within your software architecture.

Main Updates

1. Artifactory [JFrog Xray integration](#).
2. You can now restrict a user to accessing Artifactory **only through the REST API**.
3. Deprecated "Force Authentication" configuration field has been removed from [Docker repository configuration](#) that was used to enable the `docker login` command. Currently all Docker repositories support both authenticated and anonymous access according to the permission configuration making this field obsolete. This is especially useful for users representing different tools that interact with Artifactory such as CI servers, build tools, etc.
4. Artifactory now supports custom [Atlassian Crowd](#) authentication tokens.
5. Artifactory OAuth integration now supports passing in [query params](#) as part of the authorization URL.
6. AQL and Artifactory public API, have been enhanced to support reporting detailed [remote download statistics](#) for smart remote repositories.
7. When deploying archives to Artifactory using the REST API, you can specify that they should be exploded in an atomic operation through the `X-Explode-Archive-Atomic` header.
8. Removed support for deprecated `artifactory.security.useBase64` flag in `artifactory.system.properties` and as a consequence `artifactory.security.authentication.encryptedPassword.surroundChars`.

In order to trigger generation of a new encrypted password, compatible with Artifactory version 4.9.0 and above, users are required to access their user profile page and obtain a new encrypted password.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.9.1

Released: July 14, 2016

Main Updates

1. Improves performance when editing a user's details for a system with a large number of users.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.8

Released: May 23, 2016

Distribution Repository

A new repository type designed to let you push your software out to customers and users quickly and easily through JFrog Bintray. Once set up, access to Bintray is managed by Artifactory so all you need to do is put your artifacts in your distribution repository, and they automatically get pushed to Bintray for distribution.

Main Updates

1. [Distribution Repository](#)
2. Recalculation of metadata for different repository types (Ruby Gems, Npm, Bower, NuGet, Debian, YUM, Pypi, CocoaPods, Opkg) can now be triggered by users with the set of permissions assumed by Manage (i.e. Manage + Delete/Overwrite + Deploy/Cache + Annotate + Read). Previously this required admin permissions. Known limitation: triggering metadata recalculation for virtual repositories through the Artifactory UI still requires admin privileges.
3. When rewriting external dependencies for npm or Bower repositories, shorthand dependencies that are [GitHub URLs](#) will be matched by all patterns that contain "github.com"

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.8.1

Released: May 23, 2016

Change in OSS license

From version 4.8.1, Artifactory OSS is licensed under [AGPL 3.0](#) (previously LGPL 3.0).

Distribution Repositories

Added support for distribution dry run as well as support for both named and unnamed capture groups when specifying repositories and paths for distribution provides enormous flexibility in how you upload files to Bintray.

Tree Performance Improvements

Major improvement in tree loading time when working on large scale tree with thousands of entries.

Main Updates

1. Improvements to Distribution Repository
 - a. Offer enormous flexibility in how you upload files to Bintray by supporting both [named](#) and [unnamed](#) capture groups.
 - b. Added [dry run](#) option before executing distribution.
2. The [Tree Browser](#) has undergone many changes under the hood to significantly improve behavior and performance when heavily populated with many items.
3. Artifactory will now reject repository names that would conflict and create duplicate entries in the Tree Browser.
4. Resolved RubyGems error caused by version comparator method.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.8.2

Released: May 23, 2016

Main Updates

1. Conversion of the Docker manifest schema from v2 to v1 when pulling an image from a remote repository that proxies DockerHub. This issue caused Docker client below version 1.10.0 to fail pulling images uploaded with client version 1.10.0 and higher.
2. In a High Availability configuration, Artifactory fails to delete a repository if a download from the repository is in progress while the repository is being deleted.
3. Allow disabling maven auto-data calculation upon delete event. This will allow performing massive deletes.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.7

Released: March 31, 2016

Remote and Virtual Git LFS Repositories

Artifactory is the only repository manager that supports remote and virtual Git LFS repositories. Use remote repositories to easily share your binary assets between teams across your organization by proxying Git LFS repositories on other Artifactory instances or on GitHub. Wrap all your local and remote Git LFS repositories in a virtual repository allowing your Git LFS client to upload and download binary assets using a single URL.

Artifactory Query Language

AQL has two great new features!

Added a new Promotion domain. This allows you to run queries on builds according to details on their promotion status. For example, find the latest build with that has been promoted to "release" status.

In addition, we now support running queries across multiple domains, for example `items.find().include("archive.entry","artifact.module.build")`. This is especially useful since permissions can now be supported for domains which until now were available for admins only.

Authentication for Docker Repositories

We have removed the need to configure separate repositories for anonymous and authenticated users. Previously when anonymous access was enabled, Docker repositories allowed unauthenticated access, but in order to support authenticated access, using docker login for example, you had to use the "Force Authentication" flag. This limitation is now removed and anonymous users can pull and push, according to configured permissions, to all repositories, including ones checked with the "Force Authentication" flag.

As a result, the "Force authentication" checkbox in Docker repository settings is deprecated. It is currently left in the UI in a checked and immutable state for reference only, and will be removed in a future version.

NOTE: Anonymous users can continue working with existing repositories where "Force Authentication" was set to false. In a later version when this configuration will be removed, authenticated users will be able to work with those repositories as well.

Block Mismatched Mime-types in Remote Repositories

Added support to validate that a returned artifact matches the expected Mime-Type. For example, if you request a POM file but receive an HTML file, Artifactory will block the file from being cached. When such a mismatch is detected, Artifactory will return a 409 error to the client.

By default Artifactory will block HTML and XHTML Mime-Types. You can override this configuration from the **Advanced** tab in the remote repository configuration to specify the list of Mime-Types to block.

Support for AWS IAM Role with S3

There's no need to save your credentials in a text file. As another way to authenticate when using AWS S3, you can now use an IAM role instead of saving the credentials in the `$ARTIFACTORY_HOME/etc/storage.properties` file.

Main Updates

1. [Remote](#) and [virtual](#) Git LFS repositories
2. Promotion domain for AQL and cross domain queries for non-privileged users displaying any accessible field from any domain.
3. Anonymous and authenticated users can access the same docker repository.
4. [Push Docker tags to Bintray](#) directly from the Artifactory UI.

5. Support for IAM role with S3.
6. Improved node recovery mechanism when working in High Availability setup.
7. Major improvements in YUM resulting in up to 100% improvement in performance while using much less resources.
8. Block mismatched Mime-Types from being cached in remote repository.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.7.1

Released: April 4, 2016

Main Updates

1. A fix for compatibility issue with Visual Studio 2015 update

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.7.2

Released: April 4, 2016

Main Updates

1. Change PyPI repository behavior to be case insensitive and handle '-' and '_' as the same character when comparing package name.
2. To support disaster recovery in JFrog Mission Control, you can now [globally block replication](#) regardless of configuration in specific repositories.
3. Configure login link to [automatically redirect users](#) to the SAML login page.
4. AQL supports specifying [time intervals relative](#) to when queries are run.
5. Add support for the NuGet `--reinstall` command.
6. Add support for the Npm `--tag` command.
7. Add support for [AWS version](#) parameter in [Filestore Configuration](#).
8. Exposed a method to get or set user properties in [Artifactory's Public API](#).

For a complete list of changes please refer to our [JIRA Release Notes](#).

Known Issues

1. Existing PyPI packages will not available until triggering an index recalculation and setting the relevant metadata to support the new PyPI implementation.
This issue is resolved in [Artifactory 4.7.3](#) that will trigger index recalculation when upgrading from an older version for all [PyPI repositories](#).
[Related issue - RTFACT-9865](#).
 2. Upgrading to pip client 8.1.2 will introduce an issue with installing packages which contain '.' in the package name. This is due to an a change in pip client behavior that was supposed to included in 8.0.0 but only manifested in 8.1.2 due to a bug in pip client.
[Related issue - RTFACT-10133](#).
-

Artifactory 4.7.3

Released: April 17, 2016

Main Updates

1. Improved migration of existing PyPI packages to new PyPI implementation.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Known Issues

1. In case there is a conflict is artifacts resolution that can be as a result of the [block-mime types](#), or trying to resolve a maven snapshot version from a repository configured to only [handle releases](#) repository virtual repository will return a 409 (conflict) error code. Gradle clients do not handle this error gracefully and will not try to resolve artifacts from the next repository configured in the build.gradle file. This issue was resolved in [Artifactory 4.7.4](#) that reverted this improvement.
[Related issue - RTFACT-9880](#).
-

Artifactory 4.7.4

Released: April 20, 2016

Main Updates

1. Resolution from virtual repository might result in 409 error which can cause unexpected behavior if client doesn't handle error gracefully.

For a complete list of changes please refer to our [JIRA Release Notes](#).

1. Related issue - [RTFACT-9880](#).
-

Artifactory 4.7.5

Released: May 1, 2016

Main Updates

1. Added support for SHA-256 hashing for Debian packages.
2. Maven performance has been significantly improved especially when performing multiple delete operations to use significantly less resources.
3. Conversion of Docker manifest V2 schema to V1 scheme no longer requires deleting the signing key.
4. Fixed an issue with Hazelcast timing out due to file locking in Artifactory HA.
5. Added a [new REST API](#) to schedule an immediate pull, push, or multi-push replication. This replaces the [old replication REST API](#) which has been deprecated.
6. Fixed a compatibility issue with NuGet V2 requesting framework dependencies.
NOTE: You need to invoke a reindexing of your NuGet repositories once, via the UI or using the [REST API](#), for the fix to take effect.
7. Tree browser performance has been significantly improved, especially when browsing heavily annotated repositories.
8. The workflow related to disabling the internal password for externally authenticated users (for example, via LDAP) has been improved.
9. You can now deploy artifacts with multi-value properties. For existing artifacts, you can add multi-value properties or edit them [through the UI](#).

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.7.6

Released: May 9, 2016

Main Updates

1. Significantly improved performance of Maven metadata calculation on path which contains a large number of versions.
2. Disable the `/repo` repository for new Artifactory SaaS instances provisioned.
NOTE: For existing customers this change will take effect next time the `artifactory.system.properties` is re-created. This can happen when an Artifactory server is migrated to another region, or during certain maintenance operations.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.7.7

Released: May 15, 2016

Main Updates

1. Fixed PyPI compatibility issue. Package names will be normalized as described in PyPI spec (PEP 503). After upgrading an automatic reindex will be triggered for all PyPI repositories.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.6

Released: March 13, 2016

Filestore Management

This release presents great advances in filestore management with the following features:

Advanced Filestore Configuration: A new mechanism that lets you customize your filestore with any number of binary providers giving you the most flexible filestore management capability available today.

Filestore Sharding: Through filestore sharding, Artifactory offers the most flexible and reliable way to scale your filestore indefinitely.

Google Cloud Storage: Artifactory now supports another option for enterprise-grade storage with Google Cloud Storage.

AWS S3 object store: Artifactory now supports server-side encryption for AWS S3 object store.

Using Docker with AOL

From this version, there is no limitation on the number of Docker repositories you can create on AOL. You can now access Docker repositories on AOL through `{account_name}-{repo-key}.jfrog.io`

Bundled Tomcat Version

The Tomcat bundled with Artifactory has been upgraded to version 8.0.32.

Artifactory as a Bower Registry

Artifactory is now a private Bower registry as well as a repository for Bower packages. You can now use the `bower register` commands to register your packages to any remote or virtual Bower repository in Artifactory proxying your internal VCS server (e.g. Stash, Git, BitBucket).

Main Updates

This release includes the following main updates:

1. [Advanced Filestore Configuration](#).
2. [Filestore Sharding](#).
3. Support [Google Cloud Storage](#).
4. Artifactory now supports server side encryption for [AWS S3 object store](#).
5. The bundled Tomcat in which Artifactory runs has been upgraded to version 8.0.32.
6. The simple-default [repository layout](#) used in generic repositories has been updated.
7. Unlimited [Docker repositories on AOL](#).
8. Enhanced [Docker Info](#) tab showing detailed information on Docker images.
9. When [searching with the Artifactory UI](#), Artifactory performs prefix matching for search terms in all the different search modes.
10. Artifactory is now a private [Bower registry](#) as well as a repository for Bower packages.
11. The number of characters in MSSQL properties' values is now limited to 900 characters.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.6.1

Released: March 21, 2016

Main Updates

1. A fix, to accommodate a change in the Docker client, that enables re-pushing existing layers when working with Docker 1.10.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.5

Released: February 14, 2016

CocoaPods repositories

Manage your dependencies for Apple OS development through Artifactory. Artifactory supports CocoaPods with local and remote repositories.

Main Updates

1. CocoaPods Repositories.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.5.1

Released: February 18, 2016

OAuth Security Fix

This release fixes a security vulnerability related to OAuth.

YUM performance

YUM memory management had undergone additional tuning to further improve performance.

Main Updates

1. OAuth security fix.
2. YUM performance tuning.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.5.2

Released: February 28, 2016

This is a minor update that provides several bug fixes.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.4

Released: January 4, 2016

Security

Artifactory 4.4 brings more advancements to security capabilities including:

- Preventing brute force attacks at identity theft with increasingly delayed responses to repeated failed attempts at authentication, and locking out users after repeated failed login attempts.
- SSH Authentication for Git LFS and Artifactory CLI
- OAuth support for Docker client

Opkg Repositories

Artifactory is now a fully fledged Opkg repository, and generates index files that are fully compliant with the Opkg client. Create local repositories for your internal ipk packages, or proxy remote Opkg repositories. Provide GPG signatures for use with the Opkg client, and manage them using the UI or through REST API.

Trash Can

Artifactory now provides a trash can that prevents accidental deletion of important artifacts from the system. All items deleted are now stored for a specified period of time configured by the Artifactory administrator, before being permanently removed.

Main Updates

1. Local and remote [Opkg repositories](#).
2. Deletion protection with a [Trash Can](#).
3. SSH Authentication for [Git LFS](#) and [Artifactory CLI](#).
4. OAuth authentication for the [Docker Client](#). In addition, users can be granted access to their profile page using [OAuth](#) instead of having to type in their passwords.
5. Scan RubyGems to [extract their licenses](#) and display them as properties.
6. To combat unauthorized logins that use brute force, an administrator can configure [user locking](#). In addition, Artifactory also implements [temporary login suspension](#) for unauthorized REST API access.
7. Extract Docker labels and create corresponding [properties](#) on the image's manifest.json file.
8. Support for Virtual Repositories and [Inserting User Credentials](#) in Set Me Up dialogs.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.4.1

Released: January 13, 2016

Password Expiration Policy

An Artifactory administrator can now force all users to change their password periodically by enabling a password expiration policy.

Externally Authenticated Users

An Artifactory administrator can now enable users, who are authenticated using external means such as SAML SSO, OAuth or HTTP SSO, to access their profile and generate an API Key or modify their password.

Apache Reverse Proxy Configuration

In addition to NGINX, Artifactory now also provides you with the code snippet you need to configure Apache as your reverse proxy. Just feed in your reverse proxy settings, including your handling of Docker repositories, and Artifactory will generate the configuration script you can just plug into your Apache reverse proxy server.

Main Updates

1. [Password expiration policy](#)
2. Allow users authenticated by [SAML SSO](#), [OAuth](#), or [HTTP SSO](#) to access their profile and generate an API Key or modify their password.
3. [Reverse proxy configuration for Apache](#).

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.4.2

January 18, 2016

In addition to several bug fixes, this minor update fixes an issue with backward compatibility for S3 Object Store when upgrading to Artifactory v4.3 and above.

This version also presents a significant improvement in download performance.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.4.3

February 8, 2016

Basic Authentication

You can now use your API key as your password for basic authentication. This means that clients that cannot provide the API key in a header, can still be authenticated with the API key by including it instead of the password in the basic authentication credentials.

List Docker images

Using the List Docker Images REST API, you can get a list of images in your Docker repositories.

YUM Performance Improvements

Major improvements in performance when working with YUM repositories, showing up to 300% faster indexing of RPM packages.

Main Updates

This release includes the following main updates:

1. Compatibility with Docker v1.10 and the Docker Manifest v2 schema.
2. Major improvements in performance when working with YUM repositories.
3. Use your API key for [basic authentication](#).
4. [API key header](#) changed to X-JFrog-Art-API.

5. REST API to [enable or disable replication](#) tasks.
6. When authenticated externally, an admin can allow you to [access your API key](#), Bintray credentials and SSH public key without having to unlock your profile.
7. REST API to [list Docker repositories](#) using `/_catalog` end point.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.3

November 22, 2015

API Keys

You may now authenticate REST API calls with an [API key](#) that you can create and manage through your profile page or through the [REST API](#).

Package Search

Run a search based on a specific packaging format with dedicated search parameters for the selected format. Performance is improved since search is restricted to repositories with the specified format only.

Support Zone

Generate the information that our support team needs to provide the quickest resolution for your support tickets.

Dependency rewrite for Bower and NPM

Remove the dependence on external artifact resources for Bower and Npm. When downloading a Bower or Npm package, Artifactory will analyze the package metadata to evaluate if it needs external dependencies. If so, Artifactory will download the dependencies, host them in a remote repository cache, and then rewrite the dependency specification in the original package's metadata and point it to the new location within Artifactory.

Improved support for S3 object store

JFrog S3 object store now supports S3 version 4 allowing you to sign AWS with Signature v4. Multi-part upload and very large files over 5 GB in size are now also supported.

Main Updates

1. Authentication using [API keys](#).
2. [Package search](#).
3. Convenient [Support Zone](#) page for submitting support requests.
4. Improved support for [S3 object store](#) with support for S3 version 4.
5. Automatic rewrite of external dependencies for [Npm](#) and [Bower](#) repositories.
6. HTTP request object is now accessible from [Realms](#) closures in user plugins ([RTFACT-8514](#)).
7. REST API to [download a complete release](#) from VCS repositories.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.3.1

December 6, 2015

Reverse Proxy

Artifactory now provides a mechanism to generate [reverse proxy](#) configuration for NGINX. This is very helpful when using clients, like Docker, that require a reverse proxy.

Support Google Cloud Storage (GCS)

Artifactory now supports GCS as a storage provider for you Artifactory instance.

Git LFS Batch API

Artifactory now supports batch calls from the Git LFS client allowing batch multiple file uploads.

Main Updates

1. [Reverse proxy configuration generator](#)
2. [Google Cloud Storage](#)
3. [Batch file uploads for Git LFS repositories](#)

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.3.2

December 8, 2015

This is a minor update that provides several bug fixes.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.3.3

December 21, 2015

Propagating Query Params

When issuing requests through generic remote repositories in Artifactory, you may include query params in the request, and Artifactory will propagate the parameters in its request to the remote resource.

Source Absence Detection for Smart Remote Repositories

You can configure whether Artifactory should indicate when an item cached in a smart remote repository has been deleted from the repository at the remote Artifactory instance.

Main Updates

1. Query params may now be [propagated](#) to generic remote repositories
2. [Source absence detection](#) for smart remote repositories is now configurable.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.2

October 18, 2015

In addition to implementing several bug fixes and minor improvements, this release introduces a Debian Artifactory installation and Deploy to Virtual repositories .

Debian Installation

Artifactory can now be installed as a Debian package.

Deploy to Virtual

Artifactory now supports deploying artifacts to a virtual repository via REST API. All you need to do is specify a local repository aggregated within the virtual repository that will be the deploy target.

OAuth Login

Artifactory now supports login and authentication using OAuth providers. Currently, Google, Open ID and GitHub Enterprise are supported.

Artifactory Query Language (AQL)

AQL has been greatly extended to include several additional domains, including Build and Archive.Entry as primary domains, giving you much more flexibility in building queries.

Main Updates

1. Artifactory installation as a [Debian package](#)
2. Deploy artifacts to a [virtual repository](#)
3. Authentication using OAuth providers
4. [AQL](#) has been extended to include additional domains
5. Improvements to [Smart Remote Repositories](#)
6. REST API to retrieve [storage information](#)
7. Overwrite NuGet pre-release packages without delete permissions
8. Pushing Docker images to Bintray is now also supported for Docker V2 repositories

9. Several minor improvements to the UI

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.2.1

November 1, 2015

OAuth Provider

Cloud Foundry UAA is now supported as an [OAuth provider](#).

SHA256

In addition to SHA1 and MD5, [SHA2](#) checksums are now supported also.

Main Updates

1. Artifactory now supports Cloud Foundry UAA for [OAuth authentication](#).
2. Since Artifactory now fully supports the Bower client, support for older versions of Bower (below v1.5) that were using bower-art-resolver beta version is now deprecated.
3. Internet Explorer compatibility issues have been fixed.
4. Artifactory's HTTP client has been upgraded to version 4.5.
5. Automatic license analysis is now also triggered when [deploying RPMs](#).
6. SHA256 calculation is now available, on demand via the UI or via [REST API](#).
7. Several minor improvements to the UI.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.2.2

November 5, 2015

This is a minor update that provides several bug fixes.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.1

October 18, 2015

In addition to implementing several bug fixes and minor improvements, this release introduces Smart Remote Repositories and Virtual Docker Repositories.

Smart Remote Repositories

Define a repository in a remote Artifactory instance as your remote repository and enjoy advanced features such as automatic detection, synchronized properties and delete indications.

Virtual Docker Repositories

Aggregate all of your Docker repositories under a single Virtual Docker Repository, and access all of your Docker images through a single URL.

Main Updates

1. Support for [Smart Remote Repositories](#)
2. Docker enhancements with [virtual Docker repositories](#) and detailed [Docker image info](#)
3. [Context sensitive help](#)
4. [Custom message](#)
5. [Stash search results](#)
6. Enhanced AQL supporting queries in the Build domain
7. [Downloading a folder](#) from the UI and REST API
8. Ability to browse the content of tag and tar.gz files
9. Full support for [Bower](#) (out of Beta)
10. Several minor improvements to the UI

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.1.2

September 20, 2015

This is a minor update that provides a fix for clients, such as Maven, that do not use preemptive authentication.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.1.3

September 27, 2015

This is a minor update that provides a fix for Docker Login with anonymous access.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.0

JFrog is excited to announce the release of Artifactory 4.0. This release presents major changes in Artifactory providing a fresh look 'n feel with a completely revamped user interface and many other changes described below.

New User Interface

JFrog-Artifactory's user interface has been rebuilt from scratch to provide the following benefits:

- **Intuitive:** Configuration wizards for easy repository management
- **Fresh and modern:** New look and feel providing a rich user experience
- **Set Me Up:** Convenient code snippets to support simple copy/paste integration with software clients and CI tools
- **Context-focused repositories:** Repositories are optimized to calculate metadata for single package types
- **Easy access control:** Easily implement your access policies with intuitive user, group and permission management
- **Smart tables:** Group and filter any data that is presented in tables

Groovy 2.4 for User Plugins

JFrog Artifactory 4 supports Groovy 2.4 letting you enjoy the latest Groovy language features when writing [User Plugins](#).

We strongly recommend you verify that all of your current User Plugins continue to work seamlessly with this version of Groovy.

Tomcat 8 as the Container

JFrog Artifactory 4.0 only supports Tomcat 8 as its container for both RPM and standalone versions. If you are currently using a different container (e.g. Websphere, Weblogic or JBoss), please refer to [Upgrading When Using External Servlet Containers](#) for instructions on how to migrate to Tomcat 8.

System Requirements

Java

JFrog Artifactory 4.0 requires [Java 8](#)

Browsers

JFrog Artifactory 4.0 has been tested with the latest versions of Google Chrome, Firefox, Internet Explorer and Safari.

Breaking Changes

User Plugins

Some features of Groovy 2.4 are not backward compatible with Groovy 1.8. As a result, plugins based on Groovy 1.8 may need to be upgraded to support Groovy 2.4.

Multiple Package Type Repositories

JFrog Artifactory 4.0 requires you to specify a single package type for each repository. For the specified package type, Artifactory will calculate metadata and work seamlessly with the corresponding package format client. For example, a repository specified as Docker will calculate metadata for Docker images and work transparently with the Docker client.

Artifactory will not prevent you from uploading packages of a different format to any repository, however, metadata for those packages will not be calculated, and the corresponding client for those packages will not recognize the repository. For example, if you upload a Debian package to a NuGet repository, Debian metadata will not be calculated for that package, and the Debian client will not recognize the NuGet repository.

You may specify a repository as Generic and upload packages of any type, however, for this type of repository, Artifactory will not calculate any metadata and will effectively behave as a simple file system. These repositories are not recognized by clients of any packaging format.

If your system currently includes repositories that support several package types, please refer [Single Package Type Repositories](#) to learn how to migrate them to single package type repositories.

Artifactory 4.0.1

August 9, 2015

This is a minor update that provides significant enhancements to our support for Docker, additional UI improvements as well as several bug fixes.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Artifactory 4.0.2

August 12, 2015

This is a minor update that provides support for the latest Docker client 1.8.

For a complete list of changes please refer to our [JIRA Release Notes](#).

Previous Release Notes

For release notes of previous versions of JFrog Artifactory, please refer to [Release Notes](#) under the Artifactory 3.x User Guide

Pivotal Cloud Foundry JFrog Artifactory Tile Release Notes

Overview

List of modifications and versions for the PCF Tile specifically. Latest Documentation for the PCF tile is available on the [pivotal documentation page](#).

Releases

Version 1.0.32

General Availability Release 28-Jun-2016.

Features included in this release:

- * Uses JFrog Artifactory Enterprise edition 4.8.2
- * requires stemcell 3232.8

Version 1.0.1

General Availability Release 20-Jan-2016.

Features included in this release:

- * General Availability release
- * Uses JFrog Artifactory Enterprise edition 4.4.2
- * Highly available JFrog configuration out of the box with 2 VMs
- * Uses external MySQL service
- * Can use an external NFS, or a built-in NFS server which can be provisioned with PCF
- * Removed requirement for a 'healthcheck' user required in betas
- * Requires stemcell 3146.3

Artifactory 4.4.2 Stemcell 3146.3

Page Contents

- [Overview](#)
- [Releases](#)
 - [Version 1.0.32](#)

- [Version 1.0.1](#)
- [Version 0.6.10](#)
- [Version 0.6.1](#)
- [Version 0.5.8](#)

Version 0.6.10

Third beta release. Fixes a compatibility issue with versions of ops manager past 1.4.

Artifactory 4.2.0 Stemcell 3130

Version 0.6.1

Second beta release. Now has the ability to configure an NFS mount. If the tab is left blank, it will generate an NFS mount with bosh. If you are specifying an external NFS release, you should go to 'Resource Config' and set the number of NFS server instances to 0. Note that to use an external NFS, it may require no-root-squash be enabled to function correctly.

Artifactory 4.1.3

Version 0.5.8

First MVP version

Artifactory 4.1.3