

**JFROG HELP CENTER**

# JFrog Artifactory Documentation

## 1. JFrog Artifactory

### 2. Application Search

#### 2.1. Supported Search Methods

- 2.1.1. Search by Keywords
- 2.1.2. Free-Text Search
- 2.1.3. Advanced Search Using Filters
  - 2.1.3.1. Package Search
  - 2.1.3.2. Build Search
  - 2.1.3.3. Artifacts Search
    - 2.1.3.3.1. Quick Search
    - 2.1.3.3.2. Archive Search
    - 2.1.3.3.3. Artifact Package Search
    - 2.1.3.3.4. Property Search
    - 2.1.3.3.5. Checksum Search
    - 2.1.3.3.6. Trash Search
  - 2.1.3.4. Release Bundles Search

#### 2.2. Searching for Scanned Resources

#### 2.3. Searching for Pipelines

## 3. Configuring Artifactory

### 3.1. General Settings

- 3.1.1. Global Replication Blocking
- 3.1.2. Folder Download Settings
- 3.1.3. Distribution Settings
- 3.1.4. Release Lifecycle Settings
- 3.1.5. Trash Can Settings
  - 3.1.5.1. Restoring Deleted Repositories

### 3.2. HTTP Settings

- 3.2.1. Reverse Proxy Settings
  - 3.2.1.1. Use a Load Balancer in High Availability Setups
  - 3.2.1.2. Docker Reverse Proxy Settings
    - 3.2.1.2.1. Use a Reverse Proxy with Docker Client
      - 3.2.1.2.1.1. Use the Subdomain Reverse Proxy Method
      - 3.2.1.2.1.2. Use Port Bindings as Reverse Proxy Method
    - 3.2.1.2.2. Use Direct Access without a Reverse Proxy
  - 3.2.1.3. Configure a Reverse Proxy to Support mTLS
    - 3.2.1.3.1. Configure the Nginx Proxy for Self-hosted Customers
    - 3.2.1.3.2. Set up mTLS Verification and Certificate Termination on the Reverse Proxy
    - 3.2.1.3.3. Support User Identity Extraction for Request Authorization
  - 3.2.1.4. Manage Reverse Proxy via REST API
- 3.2.2. Configure Apache
  - 3.2.2.1. Apache Requirements with Reverse Proxy Configuration
  - 3.2.2.2. Set Up an Apache Server using HTTP
  - 3.2.2.3. Set Up an Apache Server using HTTPS as a Front End
  - 3.2.2.4. Configure a Custom Base URL in Artifactory for Apache
- 3.2.3. Configure NGINX
  - 3.2.3.1. Set Up the NGINX Server using HTTP or HTTPS
  - 3.2.3.2. Configure a Custom Base URL in Artifactory for NGINX

### 3.3. Import and Export

- 3.3.1. Repositories Import and Export
  - 3.3.1.1. Export Repositories
  - 3.3.1.2. Import Repositories
    - 3.3.1.2.1. Import Repository Layout
- 3.3.2. System Import and Export
  - 3.3.2.1. System Import and Export for an HA Cluster
  - 3.3.2.2. System Import and Active Repository Federations

### 3.4. Backups

- 3.4.1. Complete System Backup
- 3.4.2. Restoring a System Backup

### 3.5. Maven Indexer

### 3.6. Artifactory Security

- 3.6.1. Artifactory Security - General Settings
- 3.6.2. Artifactory Security - Certificates
  - 3.6.2.1. Add Certificates
  - 3.6.2.2. Use a Certificate with a Remote Repository
  - 3.6.2.3. Proxy a Resource that Uses a Self-Signed Certificates

### 3.7. Artifactory Configuration Descriptors

- 3.7.1. Global Configuration Descriptor
  - 3.7.1.1. Modify Descriptor Configuration Using the UI
  - 3.7.1.2. Modify Descriptor Configuration Using the REST API
  - 3.7.1.3. Bootstrap the Global Configuration
- 3.7.2. Repository Descriptor Configuration
- 3.7.3. Security Configuration Descriptor
  - 3.7.3.1. Modify Security Descriptor Using the UI
  - 3.7.3.2. Modify Security Descriptor Using the REST API
  - 3.7.3.3. Bootstrap the Security Configuration
- 3.7.4. Content Type/MIME Type
  - 3.7.4.1. MIME Type Attributes
  - 3.7.4.2. Set Content-Type During Download
- 3.7.5. Descriptor Related System Properties

### 3.8. Artifactory Configuration XSD

### 3.9. WebStart and Jar Signing

- 3.9.1. Managing Signing Keys
  - 3.9.1.1. Generate JAR Signing Keys
  - 3.9.1.2. Set Your Keystore and Keys
  - 3.9.1.3. Remove a Key Pair
  - 3.9.1.4. Configure Virtual Repositories to Sign JARs

### 3.10. RSA Key Pairs

- 3.10.1. Setting Up RSA Key Pairs
- 3.10.2. Managing RSA Key Pairs
- 3.10.3. RSA Key Pair REST API Commands

## 4. Release Lifecycle Management

### 4.1. Understanding Release Bundles v2

- 4.1.1. Types of Release Bundles
- 4.1.2. Release Bundle v2 Repositories
- 4.1.3. Release Bundles v2 and Docker Manifests

### 4.2. Release Lifecycle Management Workflow

### 4.3. Release Lifecycle Management Setup

- 4.3.1. Configure Custom Environments
- 4.3.2. Connect Distribution Edges to the Platform
- 4.3.3. Create Signing Keys for Release Bundles (v2)
- 4.3.4. Propagate Signing Keys to Edge Nodes
- 4.3.5. Configure Release Bundle Webhooks (optional)
- 4.3.6. Configure Release Bundle Promotion Webhooks (optional)

### 4.4. Manage the Release Lifecycle Using the Dashboard

- 4.4.1. Use the Release Bundle v2 Promotions Kanban Board
- 4.4.2. Use the Release Bundle v2 Distribution Board
- 4.4.3. Use the Release Bundle Version Timeline
- 4.4.4. Perform Actions on a Release Bundle Version

- 4.4.5. View Release Bundle (v2) Version Details
- 4.5. Create Release Bundles (v2)**
  - 4.5.1. Create a New Release Bundle v2 in the Platform UI
    - 4.5.1.1. Create a Release Bundle (v2) from the Builds Table
    - 4.5.1.2. Create a Release Bundle (v2) from Existing Release Bundles
    - 4.5.1.3. Create a New Version of an Existing Release Bundle
    - 4.5.1.4. Create a New Release Bundle from the Same Build
    - 4.5.1.5. Exclude Artifacts from a Release Bundle (v2)
  - 4.5.2. Create a New Release Bundle v2 using the REST API
  - 4.5.3. Release Bundle v2 Auto-Creation
  - 4.5.4. View the Contents of a Release Bundle (v2)
  - 4.5.5. View the Definition of a Release Bundle (v2)
  - 4.5.6. Change the Signing Key

## 4.6. Promote a Release Bundle (v2) to a Target Environment

- 4.6.1. Delete a Promotion

## 4.7. Distribute Release Bundles (v2)

- 4.7.1. Distribute a Release Bundle (v2) using the Platform UI
  - 4.7.1.1. Configure Path Mapping from the Last Promotion
  - 4.7.1.2. Path Mapping Guidelines for Release Bundles v2
- 4.7.2. Distribute a Release Bundle (v2) using REST APIs
- 4.7.3. Delete a Release Bundle (v2) Version from a Selected Edge Node
- 4.7.4. Delete a Release Bundle (v2) from Multiple Edge Nodes
- 4.7.5. Configure Deleted-at-Target Scraping Service

## 4.8. Distribute Release Bundles (v2) in an Air Gap Environment

- 4.8.1. Export Release Bundles v2 in the Platform UI
  - 4.8.1.1. Delete Release Bundle v2 Export
  - 4.8.1.2. Download Release Bundle v2 Export Archive
  - 4.8.1.3. Copy Release Bundle v2 Export Link
- 4.8.2. Import Release Bundles v2 in the Platform UI
- 4.8.3. Export and Import Release Bundles v2 using REST APIs

## 4.9. Download the Contents of a Release Bundle v2

## 4.10. Download All Evidence for a Release Bundle Version

### 4.11. Scan Release Bundles (v2) with Xray

### 4.12. Monitor Release Bundle v2 Versions

### 4.13. Verify Release Bundles (v2)

### 4.14. Delete a Release Bundle (v2) Version

### 4.15. Release Lifecycle Management in Federated Environments

### 4.16. Known Issues and Limitations

## 5. Evidence Management

### 5.1. Evidence Setup

### 5.2. Working with Evidence

- 5.2.1. View Evidence
  - 5.2.1.1. View the Artifact Evidence Table
  - 5.2.1.2. View the Package Evidence Table
  - 5.2.1.3. View the Build Evidence Table
  - 5.2.1.4. View the Release Bundle Evidence Graph
  - 5.2.1.5. View the Evidence Predicate
- 5.2.2. Attach External Evidence
- 5.2.3. Create Your Own DSSE

### 5.3. Understanding Evidence Files

- 5.3.1. Evidence Predicate
- 5.3.2. Evidence Payload
- 5.3.3. Evidence Envelope
- 5.3.4. Evidence Deployment Workflow
- 5.3.5. Evidence Deletion Workflow

## 6. Repository Management

### 6.1. Repository Management Overview

- 6.1.1. Quick Repository Setup
  - 6.1.1.1. Repository Naming Rules and Limitations
  - 6.1.1.2. Generic Repositories
- 6.1.2. General Resolution Order

### 6.2. Local Repositories

- 6.2.1. Configure a Local Repository
- 6.2.2. Basic Settings for Local Repositories
- 6.2.3. Advanced Settings for Local Repositories
- 6.2.4. Additional Local Repository Settings for Specific Package Types
  - 6.2.4.1. Additional Settings for Alpine Local Repositories
  - 6.2.4.2. Additional Settings for Cargo Local Repositories
  - 6.2.4.3. Additional Settings for Cocoapods Local Repositories
  - 6.2.4.4. Additional Settings for Conan Local Repositories
  - 6.2.4.5. Additional Settings for Debian Local Repositories
  - 6.2.4.6. Additional Settings for Docker Local Repositories
  - 6.2.4.7. Additional Settings for Hex Local Repositories
  - 6.2.4.8. Additional Settings for Maven/Gradle/Ivy/SBT Local Repositories
  - 6.2.4.9. Additional Settings for npm Local Repositories
  - 6.2.4.10. Additional Settings for NuGet Local Repositories
  - 6.2.4.11. Additional Settings for OCI Local Repositories
  - 6.2.4.12. Additional Settings for Opk Local Repositories
  - 6.2.4.13. Additional Settings for PHP Composer Local Repositories
  - 6.2.4.14. Additional Settings for RPM Local Repositories
  - 6.2.4.15. Additional Settings for Terraform Local Repositories

### 6.3. Remote Repositories

- 6.3.1. Configure a Remote Repository
- 6.3.2. Basic Settings for Remote Repositories
- 6.3.3. Advanced Settings for Remote Repositories
  - 6.3.3.1. Remote Credentials
  - 6.3.3.2. Network Settings for Remote Repositories
  - 6.3.3.3. Cache Settings for Remote Repositories
  - 6.3.3.4. Zipping Caches
  - 6.3.3.5. Select Property Sets
  - 6.3.3.6. Other Advanced Settings for Remote Repositories
- 6.3.4. Additional Remote Repository Settings for Specific Package Types
  - 6.3.4.1. Additional Settings for Bower Remote Repositories
  - 6.3.4.2. Additional Settings for Cargo Remote Repositories
  - 6.3.4.3. Additional Settings for Cocoapods Remote Repositories
  - 6.3.4.4. Additional Settings for Composer Remote Repositories
  - 6.3.4.5. Additional Settings for Conan Remote Repositories
  - 6.3.4.6. Additional Settings for Debian Remote Repositories
  - 6.3.4.7. Additional Settings for Docker Remote Repositories
  - 6.3.4.8. Additional Settings for Generic Remote Repositories
  - 6.3.4.9. Additional Settings for Go Remote Repositories
  - 6.3.4.10. Additional Settings for Helm OCI Remote Repositories
  - 6.3.4.11. Additional Settings for Legacy Helm Remote Repositories
  - 6.3.4.12. Additional Settings for Hex Remote Repositories
  - 6.3.4.13. Additional Settings for Maven/Gradle/Ivy/SBT Remote Repositories
  - 6.3.4.14. Additional Settings for NuGet Remote Repositories
  - 6.3.4.15. Additional Settings for Opk Remote Repositories
  - 6.3.4.16. Additional Settings for P2 Remote Repositories
  - 6.3.4.17. Additional Settings for PyPI Remote Repositories
  - 6.3.4.18. Additional Settings for RPM Remote Repositories
  - 6.3.4.19. Additional Settings for Swift Remote Repositories

# JFrog Artifactory Documentation Displayed in the header

6.3.4.20. Additional Settings for Terraform Remote Repositories
6.3.4.21. Additional Settings for VCS Remote Repositories
6.3.5. Browse Remote Repositories
6.3.6. Handling Offline Scenarios
<b>6.4. Smart Remote Repositories</b>
6.4.1. Configure a Smart Remote Repository
6.4.2. Remote List Browsing
<b>6.5. Virtual Repositories</b>
6.5.1. Configure a Virtual Repository
6.5.2. Basic Settings for Virtual Repositories
6.5.2.1. Select Repositories to Include in a Virtual Repository
6.5.2.2. Virtual Resolution Order
6.5.3. Additional Virtual Repository Settings for Specific Package Types
6.5.3.1. Additional Settings for Alpine Virtual Repositories
6.5.3.2. Additional Settings for Bower Virtual Repositories
6.5.3.3. Additional Settings for Chef Virtual Repositories
6.5.3.4. Additional Settings for Conan Virtual Repositories
6.5.3.5. Additional Settings for Conda Virtual Repositories
6.5.3.6. Additional Settings for CRAN Virtual Repositories
6.5.3.7. Additional Settings for Debian Virtual Repositories
6.5.3.8. Additional Settings for Docker Virtual Repositories
6.5.3.9. Additional Settings for Go Virtual Repositories
6.5.3.10. Additional Settings for Maven/Gradle/Ivy/SBT Virtual Repositories
6.5.3.11. Additional Settings for npm Virtual Repositories
6.5.3.12. Additional Settings for NuGet Virtual Repositories
6.5.3.13. Additional Settings for OCI Virtual Repositories
6.5.3.14. Additional Settings for P2 Virtual Repositories
<b>6.6. Federated Repositories</b>
6.6.1. Federated Topology
6.6.2. Artifactory Federation Service
6.6.2.1. Migrate to the Artifactory Federation Service
6.6.3. Setup Prerequisites for Federated Repositories
6.6.4. Federated Repository Best Practices
6.6.5. Set Up a Federated Repository
6.6.5.1. Set Up a Federated Repository Using the UI
6.6.5.2. Set Up a Federated Repository Using the REST API
6.6.5.3. Convert a Local Repository to a Federated Repository
6.6.5.3.1. Convert a Local Repository to a Federated Repository - Platform UI
6.6.5.3.2. Convert a Local Repository to a Federated Repository - REST API
6.6.5.4. Convert a Federated Repository to a Local Repository
6.6.5.5. Convert a Build-Info Repository to a Federated Repository
6.6.6. Working with Federated Repositories
6.6.6.1. Working with Federated Artifacts
6.6.6.2. Pause/Resume Federated Synchronization
6.6.6.3. Remove Members from the Federation
6.6.6.4. Federation Recovery and Auto-Healing
6.6.6.5. Perform Full Sync on Federated Repositories
6.6.6.5.1. System Properties for Full Sync File List Queries
6.6.6.6. Use the Federation Comparison Tool
6.6.6.7. Configure Federated Repositories for Bulk Mirroring and Parallel Processing
6.6.6.8. Configure In-Memory Sorting for Full Sync Operations
6.6.6.9. Geo Synchronized Topology Use Case: Setting a Federated Base URL
6.6.6.10. Change the Base URL in Federated Repositories
6.6.6.10.1. Change the Federated Base URL
6.6.6.10.2. Generate New Tokens for Federation Members
6.6.6.10.3. After Changing the Base URL
6.6.6.11. Increase the Predefined Socket Timeout for Larger Repositories
6.6.6.12. Troubleshoot Federated Member Out-of-Sync Notifications
6.6.7. Monitor Federated Repositories
6.6.7.1. Monitor Federated Repositories using the Dashboard
6.6.7.1.1. View the Status of All Repository Federations
6.6.7.1.1.1. Assign Priority to Federated Repositories
6.6.7.1.2. View the Status of a Selected Repository Federation
6.6.7.2. Monitor Federated Repositories using REST API
6.6.7.3. Monitor Federated Repositories using Open Metrics
6.6.7.3.1. Artifactory Federation Service Custom Metrics
6.6.7.3.2. Artifactory System Properties for Federated Repository Metrics
6.6.7.3.3. Federated Repository Metrics
6.6.7.3.4. Federated Repository Metrics in MBeans
6.6.7.4. Monitor Federated Repositories in the Platform UI
6.6.7.4.1. View Federated Repository Status
6.6.7.4.2. View Federation Sync Status
6.6.8. Multi-Version Support
<b>6.7. Release Bundle Repositories</b>
6.7.1. Create a Release Bundle v1 Repository
<b>6.8. Repository Replication</b>
6.8.1. Push Replication
6.8.2. Pull Replication
6.8.3. Schedule and Configure Replication Using the UI
6.8.3.1. Configure Push Replication
6.8.3.2. Add a Push Replication Target
6.8.3.3. Configure Pull Replication
6.8.4. Replicate with REST API
6.8.5. Replication Properties
6.8.6. Optimize Repository Replication Using Storage Level Synchronization Options
<b>6.9. Repository Layouts</b>
6.9.1. The Freedom of Custom Layouts
6.9.2. Bundled Layouts
6.9.3. Modules and Path Patterns used by Repository Layouts
6.9.3.1. Using Module Fields to Define Path Patterns
6.9.3.1.1. Path Pattern Tokens
6.9.3.1.2. Artifact Path Patterns
6.9.3.1.3. Descriptor Path Patterns
6.9.4. Configure Repository Layouts
6.9.4.1. Layout Configuration
6.9.4.1.1. Regular Expressions for File and Folder Integration Revision
6.9.4.2. Repository Layout Configuration
6.9.4.2.1. Local Repository Layout Configuration
6.9.4.2.2. Remote Repository Layout Configuration
6.9.4.2.3. Virtual Repository Layout Configuration
<b>6.10. Repository Support for Package Clients</b>

## 7. Package Management

### 7.1. Viewing Packages

7.1.1. Filtering the Packages List
7.1.2. Viewing Package Information
7.1.3. Viewing Xray Data on Packages
7.1.4. Viewing Package Version Information

### 7.2. Downloading Package Versions

### 7.3. Adding Packages to Projects

### 7.4. Supported Package Types

### 7.5. Alpine Linux Repositories

7.5.1. Alpine Linux Repository Structure
7.5.2. Set up an Alpine Linux Repository
7.5.2.1. Local Alpine Repositories
7.5.2.2. Remote Alpine Repositories
7.5.2.3. Virtual Alpine Repositories

# JFrog Artifactory Documentation Displayed in the header

7.5.3. Configure Alpine Linux Package Manager to work with Artifactory	
7.5.3.1. Resolve an Alpine Package	
7.5.3.2. Deploy an Alpine Package	
7.5.4. Alpine Artifact Metadata	
7.5.5. Alpine REST API Support	
<b>7.6. Ansible Repositories</b>	
7.6.1. Get Started with Ansible	
7.6.1.1. Administrator Module - Ansible Packages	
7.6.1.2. Application Module - Ansible Packages	
7.6.1.3. Types of Ansible Repositories	
7.6.2. Administer Ansible Repositories	
7.6.2.1. Create Ansible Repositories	
7.6.2.1.1. Create Local Ansible Repositories	
7.6.2.1.2. Create Remote Ansible Repositories	
7.6.2.1.3. Create Virtual Ansible Repositories	
7.6.2.2. Manage Ansible Repositories	
7.6.2.2.1. Recalculate Index of Local Ansible Repositories	
7.6.2.2.2. Edit Ansible Repositories	
7.6.2.2.2.1. Edit Local Ansible Repositories	
7.6.2.2.2.2. Edit Remote Ansible Repositories	
7.6.2.2.2.3. Edit Virtual Ansible Repositories	
7.6.2.2.3. Delete Ansible Repositories	
7.6.2.2.3.1. Delete Local Ansible Repositories	
7.6.2.2.3.2. Delete Remote Ansible Repositories	
7.6.2.2.3.3. Delete Virtual Ansible Repositories	
7.6.3. Set Me Up Ansible	
7.6.4. View Set Me Up Ansible Code Snippets	
7.6.4.1. Configure Ansible Client to Work with Artifactory	
7.6.4.1.1. Configure Ansible Client to Work with Collections using ansible.cfg File	
7.6.4.1.2. Configure Ansible Client to Work with Roles using .netrc File	
7.6.4.2. Publish Ansible Collections	
7.6.4.2.1. Publish Ansible Collections via Ansible CLI - Server in ansible.cfg File	
7.6.4.2.2. Publish Ansible Collections via Ansible CLI without ansible.cfg File - Server Explicit	
7.6.4.2.3. Deploy Ansible Collections via UI and API	
7.6.4.3. Install Ansible Collections and Roles	
7.6.4.3.1. Install Ansible Collections	
7.6.4.3.1.1. Install Ansible Collections using ansible.cfg File	
7.6.4.3.1.2. Install Ansible Collections - Server Explicit	
7.6.4.3.2. Install Ansible Roles	
7.6.4.3.2.1. Install Ansible Roles using .netrc File	
7.6.4.3.2.2. Install Ansible Roles without .netrc - Server Explicit	
7.6.5. Manage Ansible Collections and Roles	
7.6.5.1. Search Ansible Collections	
7.6.5.2. List Ansible Packages Versions/Tags	
7.6.5.3. View Ansible BuildInfo	
7.6.5.4. View Individual Ansible Collections Information	
7.6.5.5. Clean Up the Ansible Local Artifactory Cache	
<b>7.7. Bower Repositories</b>	
7.7.1. Set Up a Bower Repository	
7.7.1.1. Local Bower Repositories	
7.7.1.1.1. Deploy Bower Packages	
7.7.1.2. Remote Bower Repositories	
7.7.1.3. Virtual Bower Repositories	
7.7.1.4. Bower Repositories Advanced Configuration	
7.7.2. Use the Bower Command Line	
7.7.2.1. Use CLI with Bower Version 1.5 and above	
7.7.2.2. Use CLI with Bower Below Version 1.5	
7.7.3. Work with Bower without Anonymous Access	
7.7.4. Clean Up the Local Bower Cache	
7.7.5. Automatically Rewrite External Dependencies	
7.7.6. Register Bower Packages	
7.7.7. View Individual Bower Package Information	
<b>7.8. Cargo Package Registry</b>	
7.8.1. Set Up a Cargo Repository	
7.8.1.1. Local Cargo Repositories	
7.8.1.2. Remote Cargo Repositories	
7.8.1.3. Community Assistance To Support Cargo Virtual Repositories	
7.8.2. Resolve Cargo Packages	
7.8.2.1. Resolve Cargo Packages Using the Rust Command Line	
7.8.3. Deploy Cargo Packages	
7.8.3.1. Deploy a Package Using the Cargo Client (Recommended)	
7.8.3.2. Deploy a Cargo Package Using the UI	
7.8.3.3. Deploy a Cargo Package Using cURL	
7.8.4. View Individual Cargo Package Information	
7.8.5. Index Cargo Repositories	
7.8.5.1. Index Cargo Repositories Using Sparse Indexing	
7.8.5.2. Index Cargo Repositories Using Git Indexing	
7.8.5.3. Re-Index a Cargo Repository	
7.8.6. Cargo Package Federation Limitation and Workaround	
<b>7.9. Chef Cookbook Repositories</b>	
7.9.1. Set Up a Chef Supermarket Repository	
7.9.1.1. Set Up a Local Chef Supermarket	
7.9.1.2. Set Up a Remote Chef Supermarket	
7.9.1.3. Set up a Virtual Chef Supermarket	
7.9.2. Use the Knife Command Line	
7.9.3. Work with Artifactory and Chef without Anonymous Access	
7.9.4. Publish Chef Cookbooks	
7.9.5. Use the Berkshelf Command Line	
7.9.6. View Individual Chef Cookbook Information	
7.9.7. Search for Chef Cookbooks	
<b>7.10. CocoaPods Repositories</b>	
7.10.1. Set Up a CocoaPods Repository	
7.10.1.1. Set Up Local CocoaPods Repositories	
7.10.1.1.1. Deploy CocoaPods	
7.10.1.2. Set Up Remote CocoaPods Repositories	
7.10.1.2.1. Set Up Virtual CocoaPods Repositories	
7.10.1.3. Set Up the Pod Command Line (CDN)	
7.10.1.3.1. Use CocoaPods CDN	
7.10.1.3.1.1. Use CocoaPods CDN for Local Repositories	
7.10.1.3.1.2. Use CocoaPods CDN for Remote Repositories	
7.10.1.3.1.3. Use CocoaPods CDN for Virtual Repositories	
7.10.1.3.1.4. Identify Whether a CocoaPods Repository Uses CDN	
7.10.1.3.2. Use the cocapods-art Plugin	
7.10.1.4. Work with CocoaPods in Artifactory Without Anonymous Access	
7.10.1.5. Clean Up the Local Pod Cache	
7.10.1.6. Watch the CocoaPods Screencast	
7.10.1.7. Limitations of CocoaPods Repositories in Artifactory	
<b>7.11. Conan Repositories</b>	
7.11.1. Set Up a Conan Repository	
7.11.1.1. Local Conan Repositories	
7.11.1.2. Remote Conan Repositories	
7.11.1.3. Virtual Conan Repositories	
7.11.2. Use Conan with Artifactory	
7.11.3. View Individual Conan Package Information	
7.11.4. Conan V2 Package Support	
7.11.4.1. Conan Package V1 Backward Compatibility	
7.11.4.2. Conan Revision Indexing	
7.11.4.3. View Individual Conan V2 Package Information	
7.11.4.4. View Conan Package Revisions	
7.11.5. Conan and C/C++ Support in Xray	
7.11.5.1. Scan Conan Packages and Builds	
7.11.5.2. Scan C/C++ Builds	

# JFrog Artifactory Documentation

## Displayed in the header

### 7.12. Conda Repositories

- 7.12.1. Set Up a Conda Repository
  - 7.12.1.1. Local Conda Repositories
    - 7.12.1.1.1. Local Conda Repository Layout
  - 7.12.1.2. Remote Conda Repositories
  - 7.12.1.3. Virtual Conda Repositories
- 7.12.2. Resolve Conda Packages
  - 7.12.2.1. Resolve Conda Packages Using the UI
  - 7.12.2.2. Resolve Conda Packages Using the Conda Client
- 7.12.3. Deploy Conda Packages
- 7.12.4. View Individual Conda Package Information
- 7.12.5. Reindex a Conda Repository
- 7.12.6. Tune Conda Metadata Worker Threads

### 7.13. CRAN Repositories

- 7.13.1. Set Up a CRAN Repository
  - 7.13.1.1. Set Up Local CRAN Repositories
  - 7.13.1.2. Set Up Remote CRAN Repositories
  - 7.13.1.3. Set Up Virtual CRAN Repositories
- 7.13.2. Resolve CRAN Packages
  - 7.13.2.1. Resolve CRAN Packages Using the UI
  - 7.13.2.2. Resolve CRAN Packages Using the R Command Line
- 7.13.3. Deploy CRAN Packages
  - 7.13.3.1. Deploy a CRAN Package Using the UI
    - 7.13.3.1.1. Deploy a Binary CRAN Package
    - 7.13.3.1.2. Deploy a Source CRAN Package
  - 7.13.3.2. Deploy a CRAN Package Using curl
- 7.13.4. Apply the CRAN Official Specification to Local CRAN Repositories
- 7.13.5. View Individual CRAN Package Information
- 7.13.6. Reindex a CRAN Repository

### 7.14. Debian Repositories

- 7.14.1. Set Up a Debian Repository
  - 7.14.1.1. Set Up Local Debian Repositories
    - 7.14.1.1.1. Deploy a Debian package using the UI
    - 7.14.1.1.2. Deploy a Debian package using Matrix Parameters
    - 7.14.1.1.3. Set the Target Path for Debian
    - 7.14.1.1.4. Specify multiple Debian Layouts
    - 7.14.1.1.5. Debian Artifact Metadata
    - 7.14.1.1.6. Use Debian Metadata Validation
  - 7.14.1.2. Set Up Remote Debian Repositories
    - 7.14.1.2.1. Calculate Debian Coordinates
    - 7.14.1.3. Set Up Virtual Debian Repositories
  - 7.14.2. Sign Debian Metadata
  - 7.14.3. Work with Debian Snapshots
    - 7.14.3.1. Create Debian Snapshots
    - 7.14.3.2. Rules and Guidelines for Working With Debian Snapshots
    - 7.14.3.3. Resolve Debian Snapshots
  - 7.14.4. Add MD5 Checksum to the Debian Packages File
  - 7.14.5. Configure Authenticated Access to Debian Servers
  - 7.14.6. Acquire Debian Packages by Hash
  - 7.14.7. Use Debian InRelease Metadata Files
  - 7.14.8. Debian REST API Support
  - 7.14.9. Watch the Debian Screencast
  - 7.14.10. Limitations of Debian Repositories in Artifactory

### 7.15. Docker Registry

- 7.15.1. Docker Registries and Repositories
- 7.15.2. Set Up a Docker Repository
  - 7.15.2.1. Local Docker Repositories
  - 7.15.2.2. Remote Docker Repositories
  - 7.15.2.3. Virtual Docker Repositories
  - 7.15.2.4. Docker Reverse Proxy Settings
- 7.15.3. Promote Docker Images
- 7.15.4. Push and Pull Docker Images
  - 7.15.4.1. Docker Client Push and Pull Commands
  - 7.15.4.2. Push Multi-Architecture Docker Images to Artifactory
  - 7.15.4.3. Push Docker Images One by One
  - 7.15.4.4. Push Images in Bulk Using the Docker Buildx CLI
  - 7.15.4.5. Push Multi-Architecture Docker Images Using Docker Build
- 7.15.5. Browse Docker Repositories
  - 7.15.5.1. Docker Tag Info
  - 7.15.5.2. Docker Tag Visualization
  - 7.15.5.3. Docker Labels
  - 7.15.5.4. List Manifest Content
- 7.15.6. Search for Docker Images
- 7.15.7. List Docker Images
- 7.15.8. Deletion and Cleanup of Docker Tags and Repositories
- 7.15.9. Delete Multi-Architecture Docker Tags
- 7.15.10. View Docker Build Information
- 7.15.11. Tag Retention Logic
  - 7.15.11.1. Use Max Unique Tags
  - 7.15.11.2. Use Tag Retention
  - 7.15.11.3. Using Both Max Unique Tags and Tag Retention
  - 7.15.11.4. How Tag Retention Logic Affects Manifest Lists
  - 7.15.11.5. Limitations of Tag Retention Logic
- 7.15.12. Working with Docker Content Trust
  - 7.15.12.1. Configure Docker Notary and Docker Client
    - 7.15.12.1.1. Configure Your Docker Notary Hosts File
    - 7.15.12.1.2. Configure the Docker Notary Server
    - 7.15.12.1.3. Configure the Docker Client with Docker Content Trust
  - 7.15.12.2. Test Your Docker Content Trust Setup
- 7.15.13. Docker Advanced Topics
  - 7.15.13.1. Use a Self-signed SSL Certificate with Docker
  - 7.15.13.2. Use Your Own Certificate with Nginx
  - 7.15.13.3. Set Your Docker Credentials Manually
  - 7.15.13.4. Authenticate Docker via OAuth
- 7.15.14. Get Started With Artifactory as a Docker Registry
  - 7.15.14.1. Get Started with Docker and Artifactory Cloud
    - 7.15.14.1.1. Use Kubernetes with Artifactory Cloud
    - 7.15.14.1.2. Use Docker Client with Artifactory Cloud
    - 7.15.14.1.3. Test Your Docker and Artifactory Cloud Setup
  - 7.15.14.2. Get Started with Docker and Artifactory Self-Hosted
    - 7.15.14.2.1. Get Started with Docker Using a Reverse Proxy
      - 7.15.14.2.1.1. The Subdomain Method for Docker
      - 7.15.14.2.1.2. Nginx for Docker
      - 7.15.14.2.1.3. Apache HTTPD for Docker
      - 7.15.14.2.1.4. Test Your Docker Reverse Proxy Setup
      - 7.15.14.2.1.5. The Repository Path Method for Docker
        - 7.15.14.2.1.5.1. Configure Artifactory with Repository Path Method
      - 7.15.14.2.1.6. The Ports Method for Docker
    - 7.15.14.2.2. Get Started with Docker Without a Reverse Proxy
      - 7.15.14.2.2.1. Configure Your Docker Client Without Reverse Proxy
      - 7.15.14.2.2.2. Test Your Docker Without Reverse Proxy Setup
- 7.15.15. Migrate from Docker V1 to Docker V2
- 7.15.16. Use Docker V1
  - 7.15.16.1. Get Started with Artifactory and Docker V1
    - 7.15.16.1.1. Watch the Docker V1 Screencast
  - 7.15.16.2. Browse Docker V1 Repositories
  - 7.15.16.3. Migrate a Docker V1 repository to V2
  - 7.15.16.4. Docker V1 Deletion and Cleanups
  - 7.15.16.5. Docker V1 Advanced Topics
    - 7.15.16.5.1. Use a Self-signed SSL Certificate for Docker V1
    - 7.15.16.5.2. Docker V1 Alternative Proxy Servers
    - 7.15.16.5.3. Port Bindings for Docker V1
    - 7.15.16.5.4. Docker V1 Repository Path and Domain

# JFrog Artifactory Documentation Displayed in the header

## 7.16. Git LFS Repositories

- 7.16.1. Set Up a Git LFS Repository
  - 7.16.1.1. Set Up Local Git LFS Repositories
  - 7.16.1.2. Set Up Remote Git LFS Repositories
  - 7.16.1.3. Set Up Virtual Git LFS Repositories
  - 7.16.1.4. Set Up the Git LFS Client to Point to Artifactory
- 7.16.2. Work with Artifactory and Git LFS without Anonymous Access
- 7.16.3. Authenticate Git LFS with SSH
- 7.16.4. Use Git LFS Metadata
- 7.16.5. Use Git LFS Storage
- 7.16.6. Git LFS Quick Start Guide

## 7.17. Go Registry

- 7.17.1. Install the Go Client
- 7.17.2. Set Up a Go Repository
  - 7.17.2.1. Set Up Local Go Repositories
  - 7.17.2.2. Set Up Remote Go Repositories
    - 7.17.2.2.1. Set Up Go Repositories to Proxy Private Registries
      - 7.17.2.2.1.1. Set Up Go Proxy Mirror Repositories
      - 7.17.2.2.1.2. Proxy GitHub with Go
      - 7.17.2.2.1.3. Proxy GitHub Enterprise with Go
      - 7.17.2.2.1.3.1. Generate a Personal Access Token in GitHub Enterprise
      - 7.17.2.2.1.4. Proxy GitLab with Go
      - 7.17.2.2.1.5. Proxy Bitbucket Cloud with Go
      - 7.17.2.2.1.6. Proxy Bitbucket Server with Go
    - 7.17.2.2.2. Work with GOSUMDB
  - 7.17.2.3. Set Up Virtual Go Repositories
    - 7.17.2.3.1. Advanced Configuration for Go Repositories
- 7.17.3. Use Go with Artifactory
  - 7.17.3.1. Resolve Go Projects
    - 7.17.3.1.1. Resolve Transitive Go Dependencies Locally
  - 7.17.3.2. Build Go Packages
  - 7.17.3.3. Publish Go Projects
- 7.17.4. Limitations of Go in Artifactory

## 7.18. Hex Repositories

- 7.18.1. Get Started with Hex Repositories
- 7.18.2. Create Hex Repository
  - 7.18.2.1. Hex Repository Prerequisites
    - 7.18.2.1.1. Generate and Upload RSA Key Pair
  - 7.18.2.2. Create Hex Local Repository
  - 7.18.2.3. Create Hex Remote Repository
- 7.18.3. Set Me Up - Hex Client
  - 7.18.3.1. Download Public Key to Hex Project Folder
  - 7.18.3.2. Set Up Hex Local Repository
    - 7.18.3.2.1. Configure Mix Client with Hex Local Repository
    - 7.18.3.2.2. Deploy Hex Packages to Hex Local Repository
    - 7.18.3.2.3. Resolve Hex Packages from Hex Local Repository
  - 7.18.3.3. Set Up Hex Remote Repository - hex.pm Hosted
    - 7.18.3.3.1. Configure Hex.pm - Hosted
    - 7.18.3.3.2. Resolve Hex Package from hex.pm
    - 7.18.3.3.2.1. Resolve Hex Public Package
    - 7.18.3.3.2.2. Resolve Hex Private Package
  - 7.18.3.4. Set Up Hex Remote Repository - Self-hosted
    - 7.18.3.4.1. Configure Hex Server (Self-hosted)
    - 7.18.3.4.2. Resolve Hex Self-hosted Package
- 7.18.4. Advanced Topics - Hex Repository
  - 7.18.4.1. Hex Repository Layout and Permission
  - 7.18.4.2. Manage Hex Repository
    - 7.18.4.2.1. Recalculate Index of Hex Local Repository
    - 7.18.4.2.2. Edit Hex Repository
      - 7.18.4.2.2.1. Edit Hex Local Repository
      - 7.18.4.2.2.2. Edit Hex Remote Repository
    - 7.18.4.2.3. Delete Hex Repository
      - 7.18.4.2.3.1. Delete Hex Local Repository
      - 7.18.4.2.3.2. Delete Hex Remote Repository
  - 7.18.4.3. Manage Hex Packages
    - 7.18.4.3.1. Search Hex Packages
    - 7.18.4.3.2. List Hex Package Versions/Tags
    - 7.18.4.3.3. View Hex BuildInfo
    - 7.18.4.3.4. View Individual Hex Package Information
    - 7.18.4.3.5. Clean Up Hex Local Repository Cache
  - 7.18.4.4. Deploy Hex Packages via UI
  - 7.18.4.5. View Set Me Up Instructions - Hex Repository
  - 7.18.4.6. Hex Supported Commands

## 7.19. Hugging Face Repositories

- 7.19.1. Main Features of Hugging Face in Artifactory
- 7.19.2. Create Hugging Face Repositories
  - 7.19.2.1. Hugging Face Repository Structure
  - 7.19.2.2. Hugging Face Naming Limitations
  - 7.19.2.3. Set Up Local Hugging Face Repositories
  - 7.19.2.4. Set Up Remote Hugging Face Repositories
- 7.19.3. Set Up Hugging Face SDK To Work With Artifactory
  - 7.19.3.1. Configure Hugging Face SDK to Work With Artifactory
    - 7.19.3.1.1. Configure Hugging Face SDK With Anonymous Access
    - 7.19.3.1.2. Deploy Hugging Face Models and Datasets
    - 7.19.3.1.3. Resolve Hugging Face Models and Datasets
      - 7.19.3.1.3.1. Resolve Hugging Face Models From Private Repositories
      - 7.19.3.1.3.2. Resolve Hugging Face Models and Datasets using Libraries

## 7.20. Kubernetes Helm Chart Repositories

- 7.20.1. Helm OCI Repositories
  - 7.20.1.1. Set Up a Helm OCI Repository
    - 7.20.1.1.1. Set Up Local Helm OCI Repositories
    - 7.20.1.1.2. Set Up Remote Helm OCI Repositories
    - 7.20.1.1.3. Set Up Virtual Helm OCI Repositories
  - 7.20.1.2. Set Up Helm OCI Client To Work With Artifactory
    - 7.20.1.2.1. Configure the Helm OCI Client
      - 7.20.1.2.2. Push Helm OCI Charts
      - 7.20.1.2.3. Pull Helm OCI Charts
    - 7.20.1.3. View Individual Helm OCI Artifact Information
    - 7.20.1.4. Limitations of Helm OCI
- 7.20.2. Helm Repositories
  - 7.20.2.1. Create Helm Repositories
    - 7.20.2.1.1. Set Up Local Helm Repositories
    - 7.20.2.1.2. Set Up Remote Helm Repositories
      - 7.20.2.1.2.1. Automatically Rewrite External Dependencies for Helm Charts
    - 7.20.2.1.3. Set Up Virtual Helm Repositories
      - 7.20.2.1.3.1. Namespace Support for Helm Virtual Repositories
      - 7.20.2.1.3.2. Relative URL Support for Helm Repositories
      - 7.20.2.1.3.3. Set Multiple External Dependencies for Helm Using a List of URLs
      - 7.20.2.1.3.4. Helm Virtual Repository Index Improvements
  - 7.20.2.2. Set Up Kubernetes Helm Charts To Work With Artifactory
    - 7.20.2.2.1. Configure the Helm Client
      - 7.20.2.2.1.1. Use the JFrog Helm Client
      - 7.20.2.2.2. Deploy Helm Charts
      - 7.20.2.2.3. Resolve Helm Charts
    - 7.20.2.3. View Individual Helm Chart Information
    - 7.20.2.4. Helm Charts Partial Re-indexing
    - 7.20.2.5. Helm Enforce Layout
      - 7.20.2.5.1. Prevent Duplicate Chart Paths
      - 7.20.2.5.2. Enforce Chart Name and Version
      - 7.20.2.5.3. Activate Helm Enforce Layout
      - 7.20.2.5.4. Limitations on Helm Enforce Layout
    - 7.20.2.6. Watch the Helm ScreenCast

# JFrog Artifactory Documentation Displayed in the header

## 7.21. Machine Learning Repositories

- 7.21.1. Main Features of Machine Learning Repositories in Artifactory
- 7.21.2. Machine Learning Repositories Configuration Overview
- 7.21.3. Create Machine Learning Repositories
- 7.21.4. Machine Learning Repository Structure
- 7.21.5. Limitations of Machine Learning Repositories in Artifactory
- 7.21.6. FrogML Library
  - 7.21.6.1. File-Based Model Type
  - 7.21.6.2. Format-Aware Model Types
    - 7.21.6.2.1. CatBoost Model Type
    - 7.21.6.2.2. Hugging Face Model Type
    - 7.21.6.2.3. ONNX Model Type
    - 7.21.6.2.4. scikit-learn Model Type
    - 7.21.6.2.5. Generic Python Function Model Type
    - 7.21.6.2.6. PyTorch Model Type

## 7.22. Maven Repository

- 7.22.1. View Maven Artifacts on Artifactory
- 7.22.2. Resolve Maven Artifacts Through Artifactory
  - 7.22.2.1. Automatically Generate Maven Settings
  - 7.22.2.2. Provision Dynamic Maven Settings for Users
- 7.22.3. Manually Override the Built-in Maven Repositories
- 7.22.4. Additional Maven Setting: Mirror Any Setup
- 7.22.5. Configure Maven Authentication
  - 7.22.5.1. Force Authentication on Virtual Maven Repositories
  - 7.22.5.2. Force Maven Non-Preemptive Authentication for Local, Remote, and Virtual Repositories
- 7.22.6. Deploy Maven Artifacts
  - 7.22.6.1. Set Up Maven Distribution Management
  - 7.22.6.2. Set Up Security in Maven Settings
- 7.22.7. Watch the Maven Screencast

## 7.23. npm Registry

- 7.23.1. Create npm Repositories
  - 7.23.1.1. Set Up Local npm Registry
    - 7.23.1.1.1. npm Repository Layout
    - 7.23.1.1.2. Set Up Remote npm Registry
    - 7.23.1.1.3. Set Up Virtual npm Registry
      - 7.23.1.1.3.1. Advanced npm Virtual Repository Configuration
  - 7.23.1.2. Use the npm Command Line
    - 7.23.1.2.1. Use npm Audit Signatures
      - 7.23.1.2.1.1. npm Client Implications for npm Audit Signatures
      - 7.23.1.2.1.2. Enable ECDSA Signing in Local Repositories
      - 7.23.1.2.1.3. Identify Whether an npm Local Repository Uses ECDSA Signing
      - 7.23.1.2.1.4. Known Issues With npm Audit Signatures
  - 7.23.1.3. Set the Default npm Registry
  - 7.23.1.4. Authenticate the npm Client
    - 7.23.1.4.1. Authenticate Using npm login
    - 7.23.1.4.2. Authenticate npm Using Basic Authentication
  - 7.23.1.5. Deploy npm Packages (npm Publish)
  - 7.23.1.6. Resolve npm Packages (npm Install)
    - 7.23.1.6.1. Resolve npm Packages using dist-tags
  - 7.23.1.7. Specify the Latest Version of npm Package
  - 7.23.1.8. Work with Artifactory npm Registry without Anonymous Access
  - 7.23.1.9. Use OAuth Credentials in npm
  - 7.23.1.10. Search for npm Packages
  - 7.23.1.11. Clean Up the Local npm Cache
  - 7.23.1.12. npm Scope Packages
  - 7.23.1.13. Configure npm Client Using Login Credentials
  - 7.23.1.14. Automatically Rewrite External npm Dependencies
    - 7.23.1.14.1. Rewrite npm Workflow
  - 7.23.1.15. View Individual npm Package Information
  - 7.23.1.16. View npm Build Information
  - 7.23.1.17. Using Yarn
    - 7.23.1.17.1. Set Up Yarn with Artifactory
    - 7.23.1.17.2. Deploy npm Packages Using Yarn
    - 7.23.1.17.3. Resolve npm Packages Using Yarn
    - 7.23.1.17.4. Work with Scoped Packages in Yarn
  - 7.23.1.18. Use npm Enforce Path Layout

## 7.24. NuGet Repositories

- 7.24.1. Create NuGet Repositories
  - 7.24.1.1. Set Up Local NuGet Repositories
    - 7.24.1.1.1. NuGet Local Repository Layout
    - 7.24.1.1.2. Publish to a NuGet Local Repository
  - 7.24.1.2. Set Up Remote NuGet Repositories
  - 7.24.1.3. Set Up Virtual NuGet Repositories
- 7.24.2. Access NuGet Repositories from Visual Studio
- 7.24.3. Use the NuGet Command Line
- 7.24.4. Configure NuGet Authentication
  - 7.24.4.1. NuGet API Key Authentication
  - 7.24.4.2. NuGet Access Token Authentication
- 7.24.5. Anonymous Access to NuGet Repositories
  - 7.24.5.1. Work in NuGet Without Anonymous Access
  - 7.24.5.2. Allow Anonymous Access to NuGet Repositories
- 7.24.6. NuGet API v3 Registry Support
- 7.24.7. Configure NuGet CLI Visual Studio to Work with NuGet v3 API
- 7.24.8. NuGet SemVer 2.0 Package Support
- 7.24.9. View NuGet Build Information
- 7.24.10. Artifactory as Your Symbol Server
  - 7.24.10.1. Publish NuGet Symbol Packages to Artifactory
    - 7.24.10.1.1. Set up a Local Symbol Server Repository
    - 7.24.10.1.2. Set up a Remote Symbol Server Repository
    - 7.24.10.1.3. Set up a Virtual Symbol Server Repository
  - 7.24.10.2. Configure the NuGet CLI to Work Opposite Artifactory as the Symbol Server
  - 7.24.10.3. View Individual Symbol Package Information
  - 7.24.10.4. Debug Symbol Files in Visual Studio
- 7.24.11. Watch the NuGet Screencast

## 7.25. NVIDIA NIM Repositories

- 7.25.1. Get Started with NVIDIA NIM
- 7.25.2. Create NVIDIA NIM Remote Repository
- 7.25.3. Set Up NIM Model client
  - 7.25.3.1. Configure NIM Client
  - 7.25.3.2. Resolve NVIDIA NIM Models
- 7.25.4. Advanced Topics - NIM Repository
  - 7.25.4.1. NIM Repository Layout
  - 7.25.4.2. Manage NVIDIA NIM Remote Repositories
    - 7.25.4.2.1. Edit NVIDIA NIM Remote Repository
    - 7.25.4.2.2. Delete NVIDIA NIM Remote Repository
- 7.25.5. Manage NVIDIA NIM Models
  - 7.25.5.1. Search NVIDIA NIM Models
  - 7.25.5.2. List NVIDIA NIM Model versions/tags

## 7.26. OCI Registry

- 7.26.1. Main Features of OCI Registry in Artifactory
- 7.26.2. Set Up an OCI Repository
  - 7.26.2.1. Set Up Local OCI Repositories
  - 7.26.2.2. Set Up Remote OCI Repositories
  - 7.26.2.3. Set Up Virtual OCI Repositories
- 7.26.3. Set Up OCI Clients To Work With Artifactory
  - 7.26.3.1. Configure Podman To Work With Artifactory
  - 7.26.3.2. Configure Buildx To Work With Artifactory
  - 7.26.3.3. Configure Buildkit/buildctl To Work With Artifactory
  - 7.26.3.4. Configure WASM to OCI To Work With Artifactory
  - 7.26.3.5. Configure ORAS To Work With Artifactory
- 7.26.4. Use Referrers REST API to Discover OCI References

# JFrog Artifactory Documentation Displayed in the header

- 7.26.5. Limitations of OCI
- 7.27. Opkg Repositories
  - 7.27.1. Set Up an Opkg Repository
    - 7.27.1.1. Set Up Local Opkg Repositories
      - 7.27.1.1.1. Deploy an Opkg Package Using the UI
    - 7.27.1.2. Set Up Remote Opkg Repositories
  - 7.27.2. Configure the Opkg Client to Work with Artifactory
  - 7.27.3. Sign Opkg Package Indexes
  - 7.27.4. Opkg Authenticated Access to Servers
  - 7.27.5. REST API Support for Opkg
- 7.28. P2 Repositories
  - 7.28.1. Set Up a P2 Repository
    - 7.28.1.1. Define a Virtual P2 Repository
    - 7.28.1.2. Select Local Repositories to Add to Your Virtual P2 Repository
    - 7.28.1.3. Select Remote Repositories to Add to Your Virtual P2 Repository
    - 7.28.1.4. Create the P2 Repositories
    - 7.28.1.5. Configure Eclipse to Work With Your P2 Virtual Repository
  - 7.28.2. Allow Anonymous Access for P2
  - 7.28.3. Integrate P2 with Tycho Plugins
  - 7.28.4. Use Multiple P2 Remote Repositories with the Same Base URL
  - 7.28.5. Configure a Remote P2 Repository for GWT
- 7.29. PHP Composer Repositories
  - 7.29.1. Set Up a PHP Composer Repository
    - 7.29.1.1. Set Up Local Composer Repositories
      - 7.29.1.1.1. Deploy Composer Packages
    - 7.29.1.2. Set Up Remote Composer Repositories
      - 7.29.1.2.1. Set Composer Remote Repositories to Work With Drupal 7 and 8 Package Type
  - 7.29.1.3. Set Up Virtual Composer Repositories
  - 7.29.2. Use the Composer Command Line
  - 7.29.3. Clean Up the Local Composer Cache
  - 7.29.4. View Individual Composer Package Information
- 7.30. Pub Repositories
  - 7.30.1. Pub Repository Structure
  - 7.30.2. Set Up a Pub Repository
    - 7.30.2.1. Set Up a Local Pub Repository
    - 7.30.2.2. Set Up a Remote Pub Repository
    - 7.30.2.3. Set Up a Virtual Pub Repository
    - 7.30.2.4. Pub SemVer 2.0 Package Support
  - 7.30.3. Configure the Pub Client to Work With Artifactory
  - 7.30.4. View Individual Pub Package Information
  - 7.30.5. Re-index a Pub Repository
  - 7.30.6. REST API Support for Pub
- 7.31. Puppet Repositories
  - 7.31.1. Set Up a Puppet Repository
    - 7.31.1.1. Set Up Local Puppet Repositories
      - 7.31.1.1.1. Use Puppet Repository Layout
    - 7.31.1.2. Set Up Remote Puppet Repositories
    - 7.31.1.3. Set Up Virtual Puppet Repositories
  - 7.31.2. Use the Puppet Command Line
  - 7.31.3. Use librarian-puppet
  - 7.31.4. Use r10k for Puppet
  - 7.31.5. Deploy Modules with Puppet Publish
    - 7.31.5.1. Set Your Puppet Credentials
    - 7.31.5.2. Deploy Your Puppet Modules
  - 7.31.6. Work with Artifactory and Puppet without Anonymous Access
  - 7.31.7. Use Puppet Search
  - 7.31.8. Clean Up the Local Puppet Cache
  - 7.31.9. View Individual Puppet Module Information
  - 7.31.10. Use Puppet 4.9.1 and Below
  - 7.31.11. REST API Support for Puppet
- 7.32. PyPI Repositories
  - 7.32.1. Set Up PyPI Repositories on Artifactory
    - 7.32.1.1. Set Up Local PyPI Repositories
    - 7.32.1.2. Set Up Remote PyPI Repositories
    - 7.32.1.3. Set Up Virtual PyPI Repositories
  - 7.32.2. Set up PyPI Clients to Work with Artifactory
    - 7.32.2.1. Set Up PyPI with a Poetry Client
    - 7.32.2.2. Set Up PyPI with a Twine Client
    - 7.32.2.3. Set Up PyPI with a Pip Client
  - 7.32.3. Publish PyPI Packages Manually Using the Web UI or REST
  - 7.32.4. Search for PyPI Packages
  - 7.32.5. View Metadata of PyPI Packages
  - 7.32.6. Work with PyPI Remote Repositories with the Custom Registry Suffix
  - 7.32.7. Watch the PyPI Screencast
  - 7.32.8. Limitations of PyPI Package Registry
  - 7.32.9. Best Practices for Working with PyPI Repositories
    - 7.32.9.1. Resolve from Artifactory Using Pip
      - 7.32.9.1.1. Specify the PyPI Repository on the Command Line
      - 7.32.9.1.2. Use PyPI Credentials
      - 7.32.9.1.3. Use a Pip Configuration File
      - 7.32.9.1.4. Use a Pip Requirements File
    - 7.32.9.2. Publish PyPI Packages to Artifactory
      - 7.32.9.2.1. Use PyPI distutils or setuptools
        - 7.32.9.2.1.1. Create the \$HOME/.pyirc File
        - 7.32.9.2.1.2. Use .netrc to Upload PyPI Packages
      - 7.32.9.2.3. Upload Authenticated PyPI Packages to JFrog Artifactory
      - 7.32.9.2.4. Upload PyPI Egg and Wheel Packages
      - 7.32.9.2.5. Use PyPI Enforce Layout
      - 7.32.9.2.6. Use PyPI File Path Name Normalization
      - 7.32.9.2.7. Using Both PyPI File Path Naming Normalization and Enforce Layout
- 7.33. RPM Repositories
  - 7.33.1. Generate RPM Metadata for Hosted RPMs
  - 7.33.2. Trigger RPM Metadata Updates
  - 7.33.3. Set Up an RPM Repository
    - 7.33.3.1. Set Up Local RPM Repositories
    - 7.33.3.2. Set Up Remote RPM Repositories
    - 7.33.3.3. Set Up Virtual RPM Repositories
    - 7.33.3.4. Deploy RPM Modules to a Local Repository
  - 7.33.4. Sign RPM Metadata
  - 7.33.5. Install RPM Packages Using Yum
  - 7.33.6. Deploy RPM Packages
  - 7.33.7. Manage YUM Groups
    - 7.33.7.1. Attach a YUM Group to a Repository
    - 7.33.7.2. Use YUM Group Commands
    - 7.33.7.3. Set YUM Group Properties
  - 7.33.8. Set Up Yum Authentication
    - 7.33.8.1. Configure Proxy Server Settings for YUM
    - 7.33.8.2. Configure SSL Settings for Yum
  - 7.33.9. Use Yum Variables
  - 7.33.10. View Individual RPM Information
  - 7.33.11. Watch the RPM Screencast
- 7.34. RubyGems Repositories
  - 7.34.1. Set Up a RubyGems Repository
    - 7.34.1.1. Set Up Local RubyGems Repositories
      - 7.34.1.1.1. Use Local RubyGems Repositories
      - 7.34.1.1.2. Local RubyGems Repositories Default Files
    - 7.34.1.2. Set Up Remote RubyGems Repositories
      - 7.34.1.2.1. Use Remote RubyGems Repositories
    - 7.34.1.3. Set Up Virtual RubyGems Repositories
      - 7.34.1.3.1. Use Virtual RubyGems Repositories

# JFrog Artifactory Documentation Displayed in the header

- 7.34.2. Use the REST API for RubyGems
- 7.34.3. View RubyGems Artifact Information
- 7.34.4. Retrieve the Latest RubyGems Package Compatible With Your Ruby Versions

## 7.35. SBT Repositories

- 7.35.1. Set Up an SBT Repository
  - 7.35.1.1. Set Up Local SBT Repositories
  - 7.35.1.2. Set Up Remote SBT Repositories
  - 7.35.1.3. Set Up Virtual SBT Repositories
- 7.35.2. Configure SBT Client To Work With Artifactory
  - 7.35.2.1. Configure SBT Proxy Repositories
  - 7.35.2.2. Configure SBT Artifact Resolution
- 7.35.3. Deploy SBT Artifacts
- 7.35.4. Fork Sample SBT Project

## 7.36. Swift Registry

- 7.36.1. Swift Repository Structure
- 7.36.2. Set Up a Swift Registry
  - 7.36.2.1. Set Up a Local Swift Repository
  - 7.36.2.2. Set Up a Remote Swift Registry
  - 7.36.2.3. Set Up a Virtual Swift Registry
- 7.36.3. Configure the Swift Client to Work Opposite Artifactory
- 7.36.4. Configure the Swift Client to Work With HTTP
- 7.36.5. Search for Swift Packages
- 7.36.6. Re-Index a Swift Repository

## 7.37. Terraform/ OpenTofu Repositories

- 7.37.1. Supported Terraform Repository Types
  - 7.37.1.1. Recommended Terraform Repository Setup
- 7.37.2. Get Started with Terraform
- 7.37.3. Terraform Registry
  - 7.37.3.1. Use Terraform Module Registries
  - 7.37.3.2. Use Terraform Provider Registries
  - 7.37.3.3. Set Up a Terraform Module/Provider Registry
    - 7.37.3.3.1. Set up a Local Terraform Module/Provider Registry
    - 7.37.3.3.2. Set Up a Remote Terraform Registry
    - 7.37.3.3.3. Set Up a Virtual Terraform Registry
  - 7.37.3.4. Set Up Terraform Module/Provider Registry to Work With Artifactory
    - 7.37.3.4.1. Configure Terraform Provider Registry with Artifactory Using OpenTofu
    - 7.37.3.4.2. Configure Terraform Provider Registry with Artifactory Using Terraform Client
    - 7.37.3.4.3. Deploy Terraform Resources
      - 7.37.3.4.3.1. Deploy Terraform Providers
      - 7.37.3.4.3.2. Deploy Terraform Modules
    - 7.37.3.4.4. Resolve Terraform Resources
      - 7.37.3.4.4.1. Resolve Terraform Providers
      - 7.37.3.4.4.2. Resolve Terraform Modules
  - 7.37.3.5. Generate an Access Token for Terraform
  - 7.37.3.6. Search for Terraform Packages
  - 7.37.3.7. Calculate a Terraform Repository Metadata
  - 7.37.3.8. Publish Terraform Modules Using the JFrog CLI
    - 7.37.3.8.1. Step 1: Set Up Terraform Repositories
    - 7.37.3.8.2. Step 2: Publish Terraform Modules to Artifactory
  - 7.37.4. Terraform Backend Repository
    - 7.37.4.1. Terraform Backend Repository Structure
    - 7.37.4.2. Set Up a Local Terraform Backend Repository
    - 7.37.4.3. Set Up Terraform Backend Repository to Work With Artifactory
      - 7.37.4.3.1. Configure Terraform Backend Repository with Artifactory Using OpenTofu
      - 7.37.4.3.2. Configure Terraform Backend Repository with Artifactory Using Terraform Client
    - 7.37.4.4. Generate an Access Token for Terraform
    - 7.37.4.5. View Individual Terraform Workspace Information

## 7.38. Vagrant Repositories

- 7.38.1. Set Up a Vagrant Repository
  - 7.38.1.1. Set Up Local Vagrant Repositories
- 7.38.2. Deploy Vagrant Boxes
  - 7.38.2.1. Deploy a Vagrant Package Using the JFrog Platform WebUI
  - 7.38.2.2. Deploy a Vagrant Package Using Matrix Parameters
- 7.38.3. Provision Vagrant Boxes
- 7.38.4. Configure Authenticated Access to Vagrant Servers

## 7.39. VCS Repositories

- 7.39.1. Set Up a VCS Repository
  - 7.39.1.1. Configure VCS Repository Layout
  - 7.39.1.2. Set Up Remote VCS Repositories
  - 7.39.1.3. Use VCS To Proxy Git Providers
- 7.39.2. REST API Support for VCS
- 7.39.3. Access Private VCS Repositories

# 8. Artifact Management

## 8.1. Storing and Using Your Artifacts

## 8.2. Saving and Using Third-Party Dependencies

- 8.2.1. Configure a Package Manager Client
- 8.2.2. Browse Third-Party Dependencies
- 8.2.3. Best Practices for Managing Third-Party Dependencies

## 8.3. Working with JFrog Properties

- 8.3.1. Properties Use Cases
- 8.3.2. View and Search for Properties in the UI
- 8.3.3. Add and Delete Properties
- 8.3.4. Use Properties in JFrog
- 8.3.5. Property Sets
- 8.3.6. Using Properties in Deployment and Resolution

## 8.4. Browsing Artifacts

- 8.4.1. How to Apply Filters
- 8.4.2. How to Use Favorite Repositories
- 8.4.3. How to Use Tree Browsing
- 8.4.4. Simple Browsing
- 8.4.5. List Browsing
- 8.4.6. Remote Browsing
- 8.4.7. View Artifact Information
- 8.4.8. View Xray Data on Artifacts
- 8.4.9. Restoring Artifacts from the Trash Can
- 8.4.10. WebDAV Browsing

## 8.5. Manipulating Artifacts

- 8.5.1. Download a Folder
- 8.5.2. Move and Copy Artifacts
- 8.5.3. Delete a Single Item
- 8.5.4. Delete a Version

## 8.6. Deploying Artifacts

- 8.6.1. Deploy a Single Artifact
- 8.6.2. Deploy Multiple Files
- 8.6.3. Deploy an Artifact Bundle
- 8.6.4. Deploy to a Virtual Repository
- 8.6.5. Deploy Large Files Using Multi-Part Upload
- 8.6.6. Failed Uploads

## 8.7. Filtered Resources

- 8.7.1. Mark an Artifact as a Filtered Resource
- 8.7.2. Filtering Context
- 8.7.3. Provision Build Tool Settings
- 8.7.4. Provisioning Example

## 8.8. Using WebDAV

- 8.8.1. Authentication for davfs2 Clients
- 8.8.2. Authentication for Windows and other WebDAV clients

# JFrog Artifactory Documentation

## Displayed in the header

### 8.9. Artifactory Query Language

- 8.9.1. AQL Architecture
  - 8.9.1.1. AQL Supported Domains
- 8.9.2. AQL Syntax
- 8.9.3. Using Fields in AQL
- 8.9.4. AQL Execution
- 8.9.5. AQL Entities and Fields
- 8.9.6. Search Criteria Construction
  - 8.9.6.1. Field Criteria
  - 8.9.6.2. Properties Criteria
  - 8.9.6.3. Compounding Criteria
  - 8.9.6.4. Match Criteria on a Single Property (\$msp)
  - 8.9.6.5. Comparison Operators
  - 8.9.6.6. Using Wildcards
  - 8.9.6.7. Date and Time Format in AQL
  - 8.9.6.8. Relative Time Operators in AQL
- 8.9.7. Specify Output Fields
  - 8.9.7.1. Display All Fields
  - 8.9.7.2. Display Specific Fields
  - 8.9.7.3. Users Without Admin Privileges
  - 8.9.7.4. Filtering Properties by Key
- 8.9.8. Sorting in AQL
- 8.9.9. Display Limits and Pagination
- 8.9.10. Using AQL With Remote Repositories
  - 8.9.10.1. Examples of Searching in Remote Repositories with AQL
- 8.9.11. Using AQL With Virtual Repositories
- 8.9.12. Limiting AQL Search Results

## 9. JFrog Container Registry

### 9.1. JFrog Container Registry Documentation

### 9.2. Container Registry Functionality

### 9.3. Get Started: JFrog Container Registry

- 9.3.1. Set Up JFrog Container Registry
  - 9.3.1.1. Set up JFrog Container Registry Self-hosted Version
  - 9.3.1.2. Set up JFrog Container Registry Cloud Version
- 9.3.2. Run the Onboarding Wizard
- 9.3.3. Get Familiar with Your JFrog Container Registry
- 9.3.4. Upgrade JFrog Container Registry

## 1 | JFrog Artifactory

JFrog Artifactory is a universal DevOps solution providing end-to-end automation and management of binaries and artifacts through the application delivery process that improves productivity across your development ecosystem.

It enables freedom of choice supporting 25+ software build packages, all major CI/CD platforms, and DevOps tools you already use. Artifactory is Kubernetes ready supporting containers, Docker, Helm Charts, and is your Kubernetes and Docker registry and comes with full CLI and REST APIs customizable to your ecosystem.

### Main Features and Functionality

- **Hybrid and Multi-Cloud Environments:** You can host Artifactory on your own infrastructure, in the Cloud or use the SaaS solution providing maximum flexibility and choice.
- **Universal Binary Repository Manager:** Artifactory offers a universal solution supporting all major package formats including Alpine, Maven, Gradle, Docker, Cargo, Conda, Conan, Debian, Go, Helm, Vagrant, YUM, P2, Ivy, NuGet, PHP, NPM, RubyGems, PyPI, Bower, CocoaPods, GitLFS, Opkg, SBT, Swift, Terraform and more. For more information, see [Package Management](#).
- **Extensive Metadata:** Artifactory provides full metadata for all major package formats for both artifacts and folders. These include metadata that originates with the package itself, custom metadata added by users such as searchable properties and metadata that is automatically generated by tools such as build information and more.
- **Artifactory as Your Kubernetes Registry:** Artifactory allows you to deploy containerized microservices to the Kubernetes cluster as it serves as a universal repository manager for all your CI/CD needs, regardless of where they are running in your organization. Once you check in your App package, you can proceed to propagate and perform the build, test, promote and finally deploy to Kubernetes.
- **Massively Scalable:** Supports a variety of enterprise-scale storage capabilities including S3 Object Storage, Google Cloud Storage, Azure Blob Storage and Filestore Sharding providing unlimited scalability, disaster recovery, and unmatched stability and reliability. Accommodates large load bursts with no compromise to performance. Increase capacity to any degree with horizontal server scalability to serve any number of concurrent users, build servers and interactions.
- **High Availability:** Full active/active HA solution with live failover and non-disruptive production upgrades. For more information, see [High Availability](#).
- **Advanced CI Server Integration with Build Tools:** JFrog 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. Artifactory provides visibility into your builds through the metadata it attaches to each artifact. In this way, you can trace your container images back to their source, so you always know what's in your builds. For more information, see [Build Integration](#).
- **Custom API-Driven Automation:** Artifactory exposes an extensive REST API that provides access to its features anywhere in the development cycle. Through the API you can manage builds, repositories and artifacts, you can perform searches, apply configurations, perform maintenance tasks and more.
- **Advanced Search with Artifactory Query Language:** AQL (Artifactory Query Language) gives you unprecedented flexibility in how you search for artifacts. It offers a simple way to formulate complex queries that specify any number of search criteria, filters, sorting options and output fields.
- **Artifactory Cloud with CDN Distribution:** JFrog Artifactory Cloud with Amazon's CloudFront CDN solution allows Enterprise users to manage, control, and distribute high volumes of software distribution across multiple locations. The fully integrated advanced CDN solution removes the need to deal with the complexity of setting up a separate external CDN Caching system. For more information, see [JFrog Cloud with CDN Distribution](#).

## 2 | Application Search

The Application global **Search** provides a set of five dedicated searches for each of your resource types - **packages**, **artifacts**, **builds**, and **Release Bundles**, a dedicated Security and Compliance search for viewing scanned resource results, and a dedicated **Pipelines** search for finding your pipelines.

The active Search view changes according to the resource page you are currently viewing and displays results for the repositories on which you have permission to view. For example, the Packages Search is active when you are in the Packages page.

### Product-Specific Requirements

The following search types require the following to be installed with their associated licenses:

- Release Bundles Search: Requires JFrog Distribution to be installed with an Enterprise+ license.
- Security & Compliance Search: Requires JFrog Xray to be installed with a Pro X, Enterprise with Xray, or an Enterprise+ license.

The search field itself appears at the top of the JPD window and is always available from any window.

#### To search for an object:

1. From any page in the JDP, click the dropdown button next to the search field and choose the type of item you want to search for, for example, **Packages**.
2. Begin typing the **name** of the object you're searching for. For example, rel-build-12, then click **Enter**. Search results are displayed.

# JFrog Artifactory Documentation

## Displayed in the header

3. Click the item that matches your search. That item is loaded in the window displaying its properties.

### Search Syntax

The Application Search supports the following syntax:

- Supports a keyword-based search
- Search terms are case-sensitive
- Supports \* and ? wildcards to help you narrow down your search. For example, when searching for a package named angular, typing an\*, angula? will return angular as a result

## 2.1 | Supported Search Methods

When using the application search, or the search field on the top of the JFrog Platform available from most windows as described here [Application Search](#), you can use a number of search methods. The following are the available search methods:

The search supports the following search methods:

- [Search by Keywords](#)
- [Free-Text Search](#)
- [Advanced Search Using Filters](#)

### 2.1.1 | Search by Keywords

You can search using the application search as described in [Application Search](#) using a set of predefined keywords listed further below. Searching by keywords help make your search more effective and enables you to search for specific filetypes like packages (pkg) Repositories (repo) and more.

#### To search by keyword:

- In the search box at the top of the JPD window, type the keyword or multiple keywords from the below list within the search. For example, you can search for all Docker packages in the Docker-local repository by typing: pkgType:docker repo:docker-local.

The Advanced search filter is also based on these keywords. Selecting a filter in the Advanced search displays the keyword in the search bar.

Click here to view a list of keywords

The following table contains the full list of keywords.

Name	Description	Packages	Artifacts	Builds	Release Bundles	Security & Compliance
pkg	Searches according to a package type.  For example: pkg:docker returns results for Docker packages.					
repo	Searches according to the repository type - local, remote, virtual, or distribution.  For example: repo:docker-local type:package myApp returns results for packages in the docker-local.					
checksum	Searches according to the checksum of the package or artifact.  For example: checksum:13f2ab8fa returns results for packages or artifacts containing the checksum number.					
properties	Searches for artifacts that are annotated with properties.  For example: properties: passedQA=true,oss=true returns results for packages that have passed QA and are OSS.					
before	Searches for resources that were created before the specified date.  For example: pkg:docker before:2019-01-03 returns results for Docker packages created before this date.					
after	Searches for resources that were created after the specified date.  For example: pkg:docker after:2019-01-03 returns results for Docker packages created after this date.					

# JFrog Artifactory Documentation

## Displayed in the header

Name	Description	Packages	Artifacts	Builds	Release Bundles	Security & Compliance
cve	<p>Searches for resources that have the specified issue.</p> <p>For example: cve: CVE-2019-12 returns results for resources affected by CVE-2019-12.</p>					
license	<p>Searches for resources containing the specified license.</p> <p>For example: license:MIT returns results for resources containing the MIT license.</p>					
severity	<p>Searches for resources contain a violation with a specific severity level.</p> <p>For example: severity:medium,high returns results for resources containing security violations with Medium and High severity levels.</p>					

The following table contains dedicated Artifact keywords:

Keyword by Search Type	Supported Package Type	Supported Values
<b>Artifacts</b>		
searchType		Packages, Archive, Properties, Checksum, or Trash.
<b>Artifacts &gt; Packages</b>		
version	All Packages	Searches for artifacts according to a specific version.
category	Chef	Free text
platform	Chef and Conda	Free text
user	Conan	Free text
channel	Conan	Free text
os	Conan	Free text
arch	Conan, Conda, Debian, Opkg, RPM	Free text
buildType	Conan	Free text
compiler	Conan	Free text
priority	Debian, Opkg	Free text
maintainer	Debian, Opkg	Free text
dist	Debian	Free text
component	Debian	Free text
digest	Docker	Free text

# JFrog Artifactory Documentation

## Displayed in the header

Keyword by Search Type	Supported Package Type	Supported Values
v1	Docker	Free text
appVersion	Helm	Free text
groupId	Maven	Free text
artifactId	Maven	Free text
classifier	Maven	Free text
scope	npm	Free text
keywords	npm	Free text
boxProvider	Vagrant	Free text
Artifacts > Archive Entries		
path		Free text
classResourcesOnly		Indicates whether to search according to class resources only. Possible values: true, false Default value: false
excludeInnerClasses		Indicates whether to exclude inner classes are part of the search. Possible values: true, false Default value: false
checksumSearch		Indicates whether the search pattern needs to be searched as a checksum Possible values: true, false Default value: false

### 2.1.2 | Free-Text Search

You can search by using one or more words, terms that include wildcard symbols and Boolean expressions.

#### To search using free text:

- In the search box at the top of the JPD window, type a string that reflects the name of the item you are searching for. You can use wildcard symbols such as \* (all), and Boolean expressions. Then click **Enter**. The search results are displayed.

### 2.1.3 | Advanced Search Using Filters

You can narrow down your search results for complex searches by setting filters in the the advanced filtered search. The search contains filters that apply to all the resources, for example, name and data range, and resource-specific filters in your docker-local repository that have a specific Docker tag. For example you can search for all the Docker images in yourdocker-localrepository that have a specific Docker tag.

#### To use advanced search filters:

- From the left-hand side of the search box at the top of the **JPD window**, select a resource type, e.g. Packages.

- Click **Advanced Search**  to the right of the search field. **Advanced Search** options are displayed for that resource type.

3. Configure **Advanced Search** options for the resource type you have selected. You have the below resource types. Click a resource type to see its options:

- Package Search
- Build Search
- Artifacts Search
- Release Bundles Search

**Example**

If Packages are chosen from the object drop to the left of the search field, you can click the Advanced Search button , then have Package options. For example, a Type drop-down field appears listing package types, and you can choose Docker for example. JFrog then adds the pkgType:docker keyword to the search bar to search for a Docker.

**2.1.3.1 | Package Search**

Package Search allows you to search for packages using a number of parameters.

**To search for packages:**

1. From the left of the search field at the top of the JDP window, select **Packages**.
2. From the right of the search field, click **Advanced Search** . Advanced Search options are displayed.

3. Configure **Advanced Search** options using any of the below parameters.

**Package Search Parameters**

Parameter	Description
Type	Select the package from the Type list.
Package Name	Type the package name.
Version	Type the package version (e.g. 1.0.0). This field is enabled after you enter the Package Name.
Repositories	Limit the packages to display for the selected package type. The list displays repositories on which you have permissions.

# JFrog Artifactory Documentation

## Displayed in the header

Parameter	Description
Checksum	Type the package checksum value.

### 2.1.3.2 | Build Search

Build Search allows you to search for builds using a number of parameters.

#### To search for builds:

1. From the left of the search field at the top of the JDP window, select **Builds**.
2. From the right of the search field, click **Advanced Search** . Advanced Search options are displayed.

3. Configure **Advanced Search** options using any of the below parameters.

#### Build Search Parameters

Name	Type the build name.
From/To	Select the date range between two dates in which the build was deployed.

### 2.1.3.3 | Artifacts Search

Artifact Advanced Search offers a number of search parameters based on the type of artifact you're searching for. Each search type offers a set of input fields corresponding to the search type you selected to help narrow down your search.

#### To search for artifacts:

1. From the left of the search field at the top of the JDP window, select **Artifacts**.
2. From the right of the search field, click **Advanced Search** . Advanced Search options are displayed.

3. Select a **Type**. Advanced search options vary depending on the type of artifact you are searching for. The following artifact types are available. Click for more information on what is offered in each, then complete the fields as needed:

# JFrog Artifactory Documentation

## Displayed in the header

- Quick Search
- Archive Search
- Artifact Package Search
- Property Search
- Checksum Search
- Trash Search

4. Take **action** on an artifact if required. Several actions are available based on the resource, by clicking the arrow on the far right, as follows:

- 
- 
- View the resource contents. For example, a JSON file.
  - View the resource in the Artifact tree
  - Delete the resource

### 2.1.3.3.1 | Quick Search

Quick Search is a search type that can be selected as part Searching Artifacts. For more information on searching artifacts, see [Artifacts Search](#).

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

To run a quick search:

1. From the left of the search field at the top of the JDP window, select **Artifacts**.
2. From the right of the search field, click **Advanced Search** . Advanced Search options are displayed.

- 
- 
- 3. From the **Type** menu, select **Quick**. Quick search options are displayed.
  - 4. Configure as needed and run your search. You can specify the following parameters for your search:

Parameter	Description
Name	Type the name of the artifact. You can use ? and * as wildcards
Repositories	Select specific repositories from the Repositories list to narrow down your search

### 2.1.3.3.2 | Archive Search

# JFrog Artifactory Documentation

## Displayed in the header

### Archive Search Deprecation Notice

The Archive Search feature in JFrog Artifactory has been deprecated as of December, 31, 2021. The Archive Search feature affects Java workloads by searching through archived JARs and other files. As a result of the deprecation, your Artifactory performance will be improved due to a reduction of upload-related events and a decrease in database-related activities. Browsing JAR archives and associated files will still be available via API or the product UI.

Archive search performs a search for all the files that match your search parameters and are located in the archive folder. Typical examples are aziporjarfile, 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. For example, Org/JFrog.
Search Class Resources Only	When selected, only class resources are searched. Any other file type is filtered out of the search.
Exclude Inner Classes	When selected, inner classes are excluded from the search.
Repositories	Limits search to the specified repositories.

### View the source file

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

#### 2.1.3.3.3 | Artifact Package Search

Artifact Package Search is a search type that can be selected as part Searching Artifacts. For more information on searching artifacts, see [Artifacts 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, **Helm** search is suitable for searching through Helm Chart repositories.

### 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, Debian, and PHP Composer repositories.

### To run a package search:

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

- From the left of the search field at the top of the JDP window, select **Artifacts**.
- From the right of the search field, click **Advanced Search** . **Advanced Search** options are displayed.

3. From the **Type** menu, select **Package**. Package options are displayed.  
4. Select the package type you want to base the search on. You have the following options:

Search Type	Search Parameters
Alpine	Package name, Version, Repositories, Checksum
Bower	Package name, Version
Chef	Name, Version, Category, Platform
CocoaPods	Package name, Version
Composer	Package name, Version
Conan	Package name, Version, User , Channel, OS, Architecture, Build Type, Compiler
Conda	Package Name, Version, Arch, Platform
CRAN	Package Name, Version
Debian	File name (without the .deb extension), Distribution, Component, Architecture
Docker	Full Image Namespace, Image Tag, Image Digest
Gems	Package name, Version
Go	Package Name, Version
Helm	Helm Chart Name, Helm Chart Version, App 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

# JFrog Artifactory Documentation

## Displayed in the header

Search Type	Search Parameters
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. After searching, click the AQL Query 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, you can 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 latest in its name:

#### 2.1.3.3.4 | Property Search

Requires an Artifactory Pro license and above.

Property Search is a search type that can be selected as part Searching Artifacts. For more information on searching artifacts, see [Artifacts Search](#).

You can 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 run a property search:

1. From the left of the search field at the top of the JDP window, select **Artifacts**.
2. From the right of the search field, click **Advanced Search** . Advanced Search options are displayed.

# JFrog Artifactory Documentation

## Displayed in the header

3. From the **Type** menu, select **Property**. Property search options are displayed.
4. Click **Add Property** and in the **Key** field, type the name of the property to search for, or select one from the list provided.
5. In the **Value** field, set the value you are searching for in the specified property.
6. Add more properties to the search use the **Add Property** as required, then click **Search**. The search is run. 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 can be combined using the AND operator.

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

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

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

### 2.1.3.3.5 | Checksum Search

Checksum Search is a search type that can be selected as part Searching Artifacts. For more information on searching artifacts, see [Artifacts Search](#).

You can 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.

#### To run a checksum search:

1. From the left of the search field at the top of the JDP window, select **Artifacts**.
2. From the right of the search field, click **Advanced Search** . **Advanced Search** options are displayed.
3. From the **Type** menu, select **Checksum**. Checksum search options are displayed.
4. Copy and paste a **checksum**, then select a **repository** to search in and click **Search**. The search is run. Wildcard characters are not supported in Checksum Search, so the term entered in the search field must be valid MD5 or SHA1 value.

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

### 2.1.3.3.6 | Trash Search

# JFrog Artifactory Documentation

## Displayed in the header

Trash Search is a search type that can be selected as part Searching Artifacts. For more information on searching artifacts, see [Artifacts Search](#).

### To run a Trash search:

1. From the left of the search field at the top of the JDP window, select **Artifacts**.
2. From the right of the search field, click **Advanced Search** . Advanced Search options are displayed.

- 
3. From the **Type** menu, select **Quick**. Quick search options are displayed.
  4. Enter the artifact's name in the Query field, or select Checksum Search and type the artifact's checksum.

### 2.1.3.4 | Release Bundles Search

*Requires JFrog Distribution be installed with an Enterprise+ licence.*

The Release Bundles search enables you to search for distributable and received Release Bundles within a specified time page.

### To search for release bundle:

1. From the left of the search field at the top of the JDP window, select **Release Bundles**.
2. From the right of the search field, click **Advanced Search** . Advanced Search options are displayed.

- 
3. Configure **Advanced Search** options using any of the below parameters.
    - **Name:** Type the name of the Release Bundle
    - **Created After:** Pick a date after which the Release Bundle was created (start date)
    - **Created Before:** Pick a date before which the Release Bundle was created (end date)
  4. Click Search. The search is run and results are displayed.

## 2.2 | Searching for Scanned Resources

### Note

Requires JFrog Xray to be installed with a Pro X, Enterprise with Xray, or an Enterprise+ license.

Scanned Resources are packages, builds, Release Bundles or artifacts that have been scanned for vulnerabilities by Xray. They can be searched for when in the Xray Scans List. Searching for scanned results in the Security & Compliance search returns the scan result by resource type. The total number of scanned resources that contain an issue for the specific resource is displayed.

### To search for scanned resources:

1. In the JPD navigation panel, click **Application** .
2. Click **Xray > Scans List**. Scans list options are displayed.
3. From the right of the search field, click **Advanced Search** . Advanced Search for the Security & Compliance search are displayed.

4. Configure **Advanced Search** options using any of the below parameters.

Parameter	Description
Name	Search according to a specific issue name.
By License	Search for resources containing a specific license or unknown licences.
By Severities	Search for resources containing vulnerabilities according to severity level - High, Low, Medium, or Unknown.
Date Range	Limits search to the specified date range.

Results are displayed by resource type. Click on each tab to view the full list and click on the resource to view the issue in the dedicated resource page.

## 2.3 | Searching for Pipelines

You can search for pipelines from any area in the JPD related to Pipelines. You can use regex and \* and ? wildcards when searching for Pipelines. To see some examples, scroll down in this topic.

To search for Pipelines:

1. In the JPD navigation panel, click **Application** .
2. Click **Pipelines**, then click any option in the Pipeline menu. Pipeline options are displayed.
3. From the right of the search field at the top of the window, click **Advanced Search** . Advanced Search options for Pipelines are displayed.
4. Configure advanced search options using the below filters, then click **Search**.

You can use the following filters to narrow down your search:

Filter	Description
Name	Search according to a specific pipeline name.
Branch	Search according to a specific branch.
Triggered Before	Search for pipelines that were triggered before the specified date.
Triggered After	Search for pipelines that were triggered after the specified date.

Example 1

Example 2

Example 3

### 3 | Configuring Artifactory

You can access the General Configuration settings of JFrog Artifactory in the **Administration** module under **Artifactory | General | Settings**.

#### Saving changes

Any changes you make must be saved in order for them to take effect.

#### 3.1 | General Settings

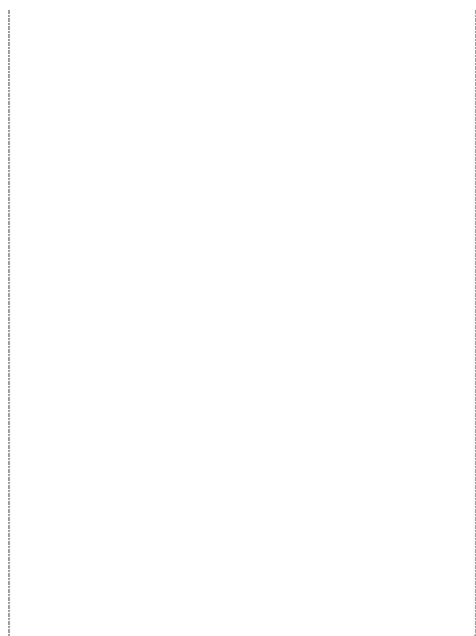
With the Artifactory General Settings, you can configure various global parameters in Artifactory.

##### Note

To view or configure these settings you need Administrative privileges. If you do not see the Administration option at the top of the JFrog instance, contact your JFrog administrator for assistance.

##### To configure the Artifactory General Settings:

1. In the **Administration** module, under **Artifactory Settings**, click **General Settings**. Artifactory General Settings are displayed.



In the **Artifactory General Settings** screen, you can configure the following general settings:

Field	Description
File Upload in UI Max Size (MB)	Maximum size allowed for files uploaded via the web interface.

# JFrog Artifactory Documentation

## Displayed in the header

Field	Description
Number of items per browser page	The maximum number of displayed items (artifacts and folders) per bulk, at the requested path level in a tree browser or native browser. If there are more items to display beyond the number you set here, a <b>Load more</b> option appears at the end of the list and when clicked displays another list of items according to this setting.
Global Offline Mode	When selected, 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).
Global Replication Blocking	Allows you to enable or disable replication globally as needed (disabled by default). For information on configuring Global Replication Blocking, see <a href="#">Global Replication Blocking</a> .
Archive Search Enabled	<p>When a new archive file is deployed to Artifactory, its content is indexed in a way that lets the user search within an archive.</p> <p>When selected, the files located within archive files are indexed to allow their content to be searchable through the "Archive Search" feature (formerly known as the "Class Search").</p> <p>It is recommended to disable this feature, if you do not use this feature as it consumes a large amount of storage.</p>
<b>Note</b>	
From Artifactory 7.28.11, archive indexing for Cloud instances has been disabled in the <code>system.yaml</code> .	
Folder Download Settings	Allow you to configure a number of options regarding how files can be downloaded from the JDP. For information about Folder download settings, see <a href="#">Folder Download Settings</a> .
Release Bundle Settings	Enable you to set the number of hours to wait before a release bundle is declared "incomplete" and eligible for cleanup from the temporary folder. For more information see <a href="#">Release Lifecycle Settings</a> .
Trash Can Settings	Enable you to determine the period that delete items should be retained before being completely removed from the system. For more information see <a href="#">Trash Can Settings</a> .

2. When finished with your configuration, click **Save**.

### 3.1.1 | Global Replication Blocking

Global Replication Blocking allows you to enable or disable replication globally as needed. Global Replication Blocking is disabled by default.

#### Note

To view or configure these settings you need Administrative privileges. If you do not see the Administration option at the top of the JFrog instance, contact your JFrog administrator for assistance.

#### To configure Global Replication Blocking:

1. In the **Administration** workspace, under **Artifactory Settings**, click **General Settings**. Artifactory General Settings are displayed.
2. Scroll down to **Global Replication Blocking**.

3. Enable Global Replication Blocking, then deselect any options as required. The following Global Replication Blocking settings are available:

Field	Description
Block Replications	Enables the use of 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.

4. When finished with your configuration, click **Save**.

### 3.1.2 | Folder Download Settings

File download settings allow you to configure a number of options regarding how files can be downloaded from the JDP, including if to enable the download of a folder and its contents or just specific files, the maximum size and maximum number of artifacts that can be downloaded under one folder, and more.

#### Note

To view or configure these settings you need Administrative privileges. If you do not see the Administration option at the top of the JFrog instance, contact your JFrog administrator for assistance.

#### To configure Folder Download Settings:

1. In the **Administration** workspace, under **Artifactory Settings**, click **General Settings**. Artifactory General Settings are displayed.
2. Scroll down to **Folder Download Settings**, then select **Enable Folder Download**.



Folder Download options are enabled.

3. Configure the required settings as follows:

Field	Description
Enable Folder Download	Must be set to enable folder download.
Enable Folder Download For Anonymous Access	Must be set to enable folder download for anonymous users.
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.

4. When finished with your configuration, click **Save**.

### 3.1.3 | Distribution Settings

The Release Lifecycle Settings area of the Artifactory General Settings screen includes settings related to JFrog Distribution and Release Bundles v1.

#### Subscription Information

This feature is supported with the **Enterprise+** license.

#### Note

To view or configure these settings you need Administrative privileges. If you do not see the Administration option at the top of the JFrog instance, contact your JFrog administrator for assistance.

1. In the **Administration** workspace, under **Artifactory Settings**, click **General Settings**. Artifactory General Settings are displayed.
2. Scroll down to **Distribution Settings**.

3. Configure the following setting:

Field	Description
Incomplete Release Bundle (v1) Cleanup Period (Hours)	The number of hours to wait before a partially distributed Release Bundle v1 is set as "incomplete" and eligible for cleanup from the temporary folder.  Set the value to <b>0</b> to disable automatic cleanup.

4. When finished with your configuration, click **Save**.

#### 3.1.4 | Release Lifecycle Settings

The Release Lifecycle Settings area of the Artifactory General Settings screen includes settings related to Release Lifecycle Management and Release Bundles v2.

##### Note

To view or configure these settings you need Administrative privileges. If you do not see the Administration option at the top of the JFrog instance, contact your JFrog administrator for assistance.

1. In the **Administration** workspace, under **Artifactory Settings**, click **General Settings**. Artifactory General Settings are displayed.
2. Scroll down to **Release Lifecycle Settings**.

3. Configure the following setting:

Field	Description
Automatically create Release Bundle (v2) when build is promoted	When set, a Release Bundle v2 version is created automatically whenever a build is promoted. By default, this option is selected.  For more information, see <a href="#">Release Bundle v2 Auto-Creation</a> .

4. When finished with your configuration, click **Save**.

#### 3.1.5 | Trash Can Settings

This topic reviews settings that enable you to control settings regarding the trash can and retention. For information regarding restoring deleted repositories see [Restoring Deleted Repositories](#). Trash can settings enable you to determine the period that delete items should be retained before being completely removed from the system.

##### Note

To view or configure these settings you need Administrative privileges. If you do not see the Administration option at the top of the JFrog instance, contact your JFrog administrator for assistance.

##### To configure Artifactory Trash Can settings:

1. In the **Administration** workspace, under **Artifactory Settings**, click **General Settings**. Artifactory General Settings are displayed.
2. Scroll down to **Trash Can Settings**.

3. Configure **Trash Can Settings** as required. The following settings are available:

Field	Description
Enable Trash Can	If set, the 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.

4. When finished with your configuration, click **Save**.

#### 3.1.5.1 | Restoring Deleted Repositories

From Artifactory release 7.41.4, two changes have been implemented in Trash Can permissions:

- Users who have **deploy or manage** permissions to any repository will be able to view the Trash Can and to view files in that repository of origin.
- Users who also have **delete** permissions to their repository will now also be able to access deleted files from that repository and to restore them without requiring admin assistance (they will not be able to view or restore any other repositories)

The contents of the Trash Can are filtered based on the repositories to which the user has permissions. This means that the user will not see or restore any other repositories unless they have permissions to that repository.

## 3.2 | HTTP Settings

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 Artifactory as Docker registry, this set up is even mandatory. To simplify configuring a reverse proxy, 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 Reverse Proxy Configuration API to manage reverse proxy configuration.

### Note

- HTTP Settings including settings for Reverse Proxy, Apache and NGINX are only available with Self-Hosted deployments. These settings do not appear when working in the SaaS deployment
- For best security, when using Artifactory behind a reverse proxy, it must be co-located on the same machine as the web server, and Artifactory should be explicitly and exclusively bound to the Proxy host
- We also recommend that you set your Custom Base URL to match your Public Server Name.

The following topics provide more information about HTTP Settings:

- [Reverse Proxy Settings](#)
- [Configure Apache](#)
- [Configure NGINX](#)

### 3.2.1 | Reverse Proxy Settings

This section describes how to configure your Artifactory to work with a Reverse Proxy server.

#### NGINX and Apache Reverse Proxy 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 `thesites-enabled` directory. For more information, see [Configuring NGINX](#).
- [To use Apache as a reverse proxy server](#), you need to have a set of modules installed and activated. For more information see [Apache Requirements with Reverse Proxy Configuration](#). For additional guidance on configuring Apache reverse proxy support, see [Configure Apache](#).

To configure reverse proxy settings for Artifactory:



1. In the navigation panel on the left-hand side of the screen, click **Administration**.

# JFrog Artifactory Documentation

## Displayed in the header

2. In the **General** pane, click **HTTP Settings**. The **HTTP Settings** screen appears.
3. Under **Server Provider**, select a service provider, for example **Nginx**. Reverse Proxy settings are displayed.

4. Configure settings as required. Refer to the parameters in the table **Table 1, "Reverse Proxy Settings"** below, then click **Save**.
5. Click **Download** to generate a reverse proxy snippet (configuration file).
6. Place the configuration file in the right place under your reverse proxy server installation and reload the configuration.

For example, Nginx Server Provider: Place the configuration file (rt.conf) in the /etc/nginx/conf.d.

### Best practice

When using a reverse proxy, it is recommended to pass the **X-JFrog-Override-Base-Url** header as follows:

#### For NGINX:

```
proxy_set_header X-JFrog-Override-Base-Url $http_x_forwarded_proto://$host:<server port>
```

#### For Apache:

```
RewriteCond %{REQUEST_SCHEME} (.*)
RewriteRule (.*)
[ E=my_scheme:%1]
[ ...]
RequestHeader set X-JFrog-Override-Base-Url %{my_scheme}://<server_name>
```

Table 1. Reverse Proxy Settings

Field	Description
Server Provider	Set the server provider type: Embedded Tomcat, Nginx and Apache.
Internal Hostname	The internal server name for Artifactory which will be used by the web server to access the Artifactory machine. If the web server is installed on the same machine as Artifactory you can use localhost, otherwise use the IP or hostname.
Internal Artifactory Port	Direct access to Artifactory for REST API and downloads. This can be configured from the Artifactory System YAML file.
Internal Router Port	Access to the JFrog Platform services REST API and web UI. This can be configured from the Artifactory System YAML file.
Public Server Name	The server name which will be publicly used to access Artifactory within the organization.
Use HTTP	When set, Artifactory will be accessible via HTTP at the corresponding port that is set.
Use HTTPS	When set, Artifactory will be accessible via HTTPS at the corresponding port that is set.

# JFrog Artifactory Documentation

## Displayed in the header

Field	Description
HTTP Port	The port for access via HTTP. The default value is 80.
HTTPS Port	The port for access via HTTPS. The default value is 443.

### 3.2.1.1 | Use a Load Balancer in High Availability Setups

When configuring Artifactory in a High Availability (HA) setup, Artifactory will automatically adjust the provided settings to include a Load Balancing section within the HTTP settings, allowing for proxying more than one Artifactory instance.

### 3.2.1.2 | Docker Reverse Proxy Settings

This section reviews Reverse Proxy settings related to Docker. When using Artifactory as a self-hosted private Docker registry, the Docker client can access Artifactory through a reverse proxy or directly through Artifactory's embedded Tomcat.

#### JFrog Artifactory Cloud Docker Registries

Note that accessing an Artifactory Docker registry on a JFrog Artifactory Cloud installation does not use a reverse proxy since it is external to your organization.

The following topics review subjects related to Docker Reverse Proxy settings:

- [Use a Reverse Proxy with Docker Client](#)
- [Use Direct Access without a Reverse Proxy](#)

For information on accessing Docker repositories without a reverse proxy, see [Use Direct Access without a Reverse Proxy](#).

#### 3.2.1.2.1 | Use a Reverse Proxy with Docker Client

The Docker client can access Artifactory through a reverse proxy using the following methods:

- Subdomain Method (recommended)
- Ports Method

For each of these methods, 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.

#### Configuring Artifactory as your Docker Registry

This section describes how to obtain your reverse proxy configuration according to whether you are using the subdomain method or port bindings.

##### 3.2.1.2.1.1 | Use the Subdomain Reverse Proxy Method

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 Settings in HTTP Settings	Corresponding HTTP Settings in Docker Repository Advanced Configuration

##### 3.2.1.2.1.2 | Use Port Bindings as Reverse Proxy Method

# JFrog Artifactory Documentation Displayed in the header

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, 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 [Virtual Docker 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 Settings in HTTP Settings	Corresponding HTTP Settings in Docker Repository Advanced Configuration

### 3.2.1.2.2 | Use Direct Access without a Reverse Proxy

To access your Docker repositories Without a Reverse Proxy, you should select **Repository Path** as the Docker Access Method in the Docker Setting Panel of the HTTP Settings screen.

Docker Settings in HTTP Settings	Corresponding HTTP Settings in Docker Repository Advanced Configuration

### 3.2.1.3 | Configure a Reverse Proxy to Support mTLS

From Artifactory release 7.38.4, you can also authenticate users using mTLS. To do so will require a reverse proxy and some setup on the front reverse proxy (Nginx).

#### Reverse Proxy for Cloud Customers

To configure a reverse proxy to support mTLS in the Cloud, you will need to contact JFrog Support to set this up for you.

#### To configure Reverse Proxy to Support mTLS Flow:

1. The client sends a request to the JFrog Platform.
  2. If the request includes a client certificate:
    - a. The JFrog Platform will authenticate the client certificate using the configured trusted certificates and verify that the certificate has not been revoked. If the client certificate is authenticated successfully, the procedure will continue; otherwise it is blocked.
    - b. The JFrog Platform will then try to extract the user identity from the client certificate.
- If the user identity was extracted successfully, the procedure will continue; otherwise it will fall back to relying on additional user authentication information (e.g., basic credentials, bearer token).

# JFrog Artifactory Documentation

## Displayed in the header

### Note

If the JFrog Platform is configured to require client certificates, then the request will be blocked; otherwise it will continue with the existing authorization mechanisms without mTLS.

#### 3.2.1.3.1 | Configure the Nginx Proxy for Self-hosted Customers

To configure the Nginx proxy, you will need to set this configuration in the Nginx configuration file, and to set the Platform configuration via the `system.yaml` file. See [Configuring Nginx](#) for details.

1. Set up mTLS by providing a trusted certificate for the JFrog Platform to trust. The trusted certificate can be either the actual client certificate to trust or a CA certificate - to trust any certificate signed by it (preferred).
2. Then, use the client certificates to authenticate API requests with the JFrog Platform (requests from untrusted client certificates will be blocked).
3. You will be able to revoke certificates by revoking (removing) the provided trusted certificate(s). You can also revoke a specific client certificate without requiring revoking the trusted CA certificate using the OCSP protocol.

### Note

You will be able to revoke certificates by revoking (removing) the provided trusted certificate(s). You can also revoke a specific client certificate without requiring revoking the trusted CA certificate.

#### 3.2.1.3.2 | Set up mTLS Verification and Certificate Termination on the Reverse Proxy

Setting up mTLS requires you to first set up mTLS verification and certificate termination on your reverse proxy. For example, using **Nginx** should include something like this:

```
ssl_verify_client      optional;
ssl_verify_depth       2;
ssl_client_certificate /path/to/client-ca.crt;
...
proxy_set_header X-JFrog-Client-Cert "";
proxy_set_header X-JFrog-Client-Cert $ssl_client_escaped_cert;
```

### Note

The `client-ca.crt` above is an example of a single file with all trusted client CA certificates.

The reverse proxy should be responsible (by configuration) for:

- **Always removing the custom header from all incoming requests**, to prevent a malicious user from adding such header on their own, tricking the platform to accept the header as an authentication and authorization mechanism
- **Adding to the request the custom header with the client certificate only for requests that were successfully mTLS-verified**

After setting your reverse proxy, when a request is performed with mTLS, upon successful verification, the reverse proxy must add a custom header with the client certificate in PEM format (refer to the `proxy_set_header X-JFrog-Client-Cert` in the code example above).

Note that you can also set up your **own custom header** instead of `X-JFrog-Client-Cert`. If you choose to do so, you will need to set the same header via the `header-name` in the `system.yaml` file (see configuration example below) for the JFrog Platform to use the same header.

#### 3.2.1.3.3 | Support User Identity Extraction for Request Authorization

You will be able to use your client certificate to authenticate and authorize requests in the JFrog Platform, without the need to send additional credentials, as long as the client certificate embeds the user identity.

To enable user-based access for client authenticating with mTLS, you can have your certificate contain a username, and the JFrog Platform will only allow access to resources to which that user has permissions.

For example, your certificate's subject might look something like this, whereby `Subject: C=IL, L=Netanya, O=Maldin, OU=DO, CN=myuser@ jfrog.com`, and where the username you are after is "myuser" from the Subject's CN. In this case, you can set the Access Configuration YAML File regexp to look something like this:

```
security:
  authentication:
    mtls: # Mutual-TLS authentication configuration
    enabled: true          # if true then mTLS is enabled
    extraction-regex: [^@]+(?=\d{0,}@) # regular expression used to extract the username from the certificate's subject CN
```

### From Version 7.77.x

Set the `access.config.latest.yml` to look something like this:

```
# To change the configuration in an Access service, follow these instructions:
# 1. Find the value you want to update
# 2. Set the value in the access.config.latest.yml file under [$JFROG_HOME]/artifactory/var/etc/access
# 3. Change access.config.latest.yml to access.config.import.yml
# 4. Restart Access
```

```
security:
  authentication:
    mtls: # Mutual-TLS authentication configuration
    enabled: true          # if true then mTLS is enabled
    extraction-regex: [^@]+(?=\d{0,}@) # regular expression used to extract the username from the certificate's subject CN
```

You can also set your regular expression to be whatever you need, in order to parse the username as it is defined in the JFrog Platform from the subject's CN attribute.

#### 3.2.1.4 | Manage Reverse Proxy via REST API

# JFrog Artifactory Documentation

## Displayed in the header

Artifactory also supports managing reverse proxy configuration through the REST API using the following endpoints:

Endpoint	Description
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.

### 3.2.2 | Configure Apache

This section reviews how to configure an Apache Server as the Artifactory front end.

#### Note

- Reverse Proxy using Apache requires you have certain modules activated and installed. For a list of these modules see [Apache Requirements with Reverse Proxy Configuration](#).
- Some features in the Apache configuration are only supported from Apache HTTP Server v2.4.

Configuring Apache support for Artifactory can include the following steps:

- Set Up an Apache Server using HTTP
- Set Up an Apache Server using HTTPS as a Front End
- Configure a Custom Base URL in Artifactory for Apache

#### 3.2.2.1 | Apache Requirements with Reverse Proxy Configuration

This topic lists the modules required when using Apache as your reverse proxy server. For more information on configuring Apache as your reverse proxy server, see [Configure Apache](#).

To use Apache as your reverse proxy server, make sure you have the following modules installed and activated:

- proxy\_http
- rewrite
- deflate
- headers
- proxy\_balancer
- proxy\_connect
- proxy\_html
- ssl
- lbmethod\_byrequests
- slotmem\_shm
- proxy

#### 3.2.2.2 | Set Up an Apache Server using HTTP

##### Apache HTTP server using AJP Protocol

From Artifactory version 7.0, the AJP connector is not supported.

You can set up Apache HTTP Server as a front end to Artifactory using the HTTP protocol.

Client -----> HTTPD -----> Artifactory  
HTTP            HTTP

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.

The sample virtual host assumes that Artifactory listens on port 8081 and all other services and UI are available on port 8082.

##### 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://yourdomain.com/`.

##### Set a PassReverse Directive on Apache

```
<VirtualHost *:80>
  ServerName yourdomain.com
  ServerAlias *.yourdomain.com
```

# JFrog Artifactory Documentation

## Displayed in the header

```
ServerAdmin server@admin

## Application specific logs
## ErrorLog ${APACHE_LOG_DIR}/yourdomain.com-error.log
## CustomLog ${APACHE_LOG_DIR}/yourdomain.com-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 ^(/)?$ /ui/ [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-JFrog-Override-Base-Url %{my_scheme}e://yourdomain.com:%{my_server_port}e

ProxyPassReverseCookiePath /
ProxyRequests off
ProxyPreserveHost on
ProxyPass "/artifactory/" http://<artifactory-ip>:8081/artifactory/ connectiontimeout=5 timeout=2400
ProxyPassReverse "/artifactory/" http://<artifactory-ip>:8081/artifactory/
ProxyPass "/" http://<artifactory-ip>:8082/ nocanon connectiontimeout=5 timeout=2400
ProxyPassReverse "/" http://<artifactory-ip>:8082/
</VirtualHost>
```

### 3.2.2.3 | Set Up an Apache Server using HTTPS as a Front End

You can set up Apache with SSL (HTTPS) as a front end to Artifactory using the HTTP protocol.

Client -----> HTTPD -----> Artifactory  
HTTPS            HTTP

#### Set SSL/TLS on Apache

```
<VirtualHost *:443>
    SSLEngine on
    SSLCertificateFile path/to/yourdomain.com.crt
    SSLCertificateKeyFile path/to/yourdomain.com.key
    SSLProxyEngine on

    ## Additional reverse proxy directives
</VirtualHost>
```

### 3.2.2.4 | Configure a Custom Base URL in Artifactory for Apache

When using an HTTP proxy, the links produced by Artifactory, as well as certain redirects might contain the wrong port and use http instead of https.

Therefore, you must configure a custom base URL as follows:

1. On the **Admin** tab select **Configuration | General** Custom Base URL field.
2. Set the **Custom Base URL** field to the value used to contact Artifactory

For example: <https://yourdomain.com>

Please refer to General System Settings for more details about configuring the base URL.

### 3.2.3 | Configure NGINX

This section describes how to set up Nginx as the Artifactory front end. 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.

# JFrog Artifactory Documentation

## Displayed in the header

Configuring Nginx support for Artifactory includes the following steps:

- Set Up the NGINX Server using HTTP or HTTPS
- Configure a Custom Base URL in Artifactory for NGINX

### 3.2.3.1 | Set Up the NGINX Server 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.

#### Seeing timeouts on large file uploads?

Up-to-date versions of Nginx have `proxy_request_buffering` enabled by default. With request buffering enabled, Nginx buffers the entire client payload prior to sending it to the Artifactory upstream.

As a result, you might see a certain stall that could range from several seconds to several minutes depending on your network performance, after the client finishes transmitting all the bytes to Nginx, as Nginx would be busy transmitting all the bytes to the Artifactory upstream at once. If this stall ranges more than a few seconds, you may start seeing client request timeouts depending on which client is used. If you are seeing timeouts, consider turning off `proxy_request_buffering` with the following directives:

```
proxy_request_buffering off;  
proxy_http_version 1.1;
```

#### Configure nginx to use HTTP or HTTPS

```
## add ssl entries when https has been set in config  
ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;  
ssl_certificate      /etc/nginx/ssl/yourdomain.com.crt;  
ssl_certificate_key /etc/nginx/ssl/yourdomain.com.key;  
ssl_session_cache shared:SSL:1m;  
ssl_prefer_server_ciphers on;  
## server configuration  
server {  
    listen 443 ssl;  
    listen 80 ;  
  
    server_name yourdomain.com;  
  
    if ($http_x_forwarded_proto = '') {  
        set $http_x_forwarded_proto $scheme;  
    }  
    ## Application specific logs  
    ## access_log /var/log/nginx/yourdomain.com-access.log timing;  
    ## error_log /var/log/nginx/yourdomain.com-error.log;  
    rewrite ^/$ /ui/ redirect;  
    rewrite ^/ui$ /ui/ redirect;  
    chunked_transfer_encoding on;  
    client_max_body_size 0;  
    location / {  
        proxy_read_timeout 2400s;  
        proxy_pass_header Server;  
        proxy_cookie_path ~*^/.*/;  
        proxy_pass http://<artifactory-ip>:8082;  
        proxy_next_upstream error timeout non_idempotent;  
        proxy_next_upstream_tries 1;  
        proxy_set_header X-JFrog-Override-Base-Url $http_x_forwarded_proto://$host:$server_port;  
        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;  
  
        location ~ ^/artifactory/ {  
            proxy_pass http://<artifactory-ip>:8081;  
        }  
    }  
}
```

#### 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.

#### Warning

Although binary caching can be enabled on Nginx reverse proxies, it is not supported by Artifactory at this time. There are known stability problems when a cache layer is set up between a client and Artifactory, such as stale metadata and upload issues. Consider setting up a Smart Remote Repository if you wish to cache artifacts.

### 3.2.3.2 | Configure a Custom Base URL in Artifactory for NGINX

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 Base URL field.
2. Set the **Custom Base URL** field to the value used to contact Artifactory

For example: `https://yourdomain.com`

# JFrog Artifactory Documentation

## Displayed in the header

Please refer to General Configuration for more details about configuring the base URL.

### 3.3 | Import and Export

Artifactory supports the import and export of data at two levels:

- **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.
- **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).

#### 3.3.1 | Repositories Import and Export

##### Note

Importing repositories is supported for all users whereas exporting repositories is only available for non-SaaS users.

To access import and export of repositories, in the **Administration** module, select **Artifactory | Import & Export | Repositories**. The following topics provide instructions on exporting and importing repositories:

- Export Repositories
- Import Repositories

##### 3.3.1.1 | Export Repositories

Exporting repositories enables you to export data and metadata stored in that repository. You may need to do this as part of moving store and metadata between repositories and for batch population of a repository.

**To export a repository:**

1. In the navigation panel on the left-hand side of the screen, click **Administration** .
2. In the General section of this screen, click **Settings**. The **Artifactory General Settings** screen appears.
3. In the **Import & Export** pane, click **Repositories**. The **Repositories Import and Export** screen appears.

4. Configure Export parameters, as shown below, then click **Export**. The file is exported.

Field	Description
Target Local Repository	You can specify a single repository to export or <b>All Repositories</b>
Export Path on Server	The export target directory on your server
Exclude Metadata	When set, repository metadata is 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

# JFrog Artifactory Documentation

## Displayed in the header

Field	Description
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> <p><b>Monitoring the log</b></p> <p>You can monitor the log in the System Logs page.</p>

### 3.3.1.2 | Import Repositories

You can import repositories previously exported as described in [Export Repositories](#). This is typically done when moving store and metadata between repositories and as part of batch operations.

#### Note

- Imported repositories have a required layout. For information on the required layout see [Import Repository Layout](#)
- To disable the system import option, set the `artifactory.system.import.enabled` property in the `artifactory.system.properties` file to false
- To view or configure these settings you need Administrative privileges. If you do not see the Administration option at the top of the JFrog instance, contact your JFrog administrator for assistance.
- To access import and export of your entire system, in the [Administration](#) module, select **Import & Export | System**

#### To import a previously exported repository:

- In the [Administration](#) workspace, under **Artifactory Settings**, click **Repository Import/Export**. The **Import Repositories** screen appears.

All users can import repositories from by zipping a repository and uploading it to Artifactory. Additionally, Self-Hosted users have the option of importing from a server-side folder.

- Complete the details as listed below, then click **Import**. The repository content is imported.

Field	Description
Target Local Repository	You can specify a single repository to import, or <b>All Repositories</b> . 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. Note: This option is only available for Self-Hosted users.
Repository Zip File	Applicable when importing repo from zip file. Drop the repository zip file
Exclude Metadata	When set, repository metadata are excluded from the import
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> <p><b>Monitoring the log</b></p> <p>You can monitor the log in the System Logs page.</p>

#### 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.

# JFrog Artifactory Documentation

## Displayed in the header

### 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.

#### 3.3.1.2.1 | Import Repository Layout

This topic describes the required layout for repositories when importing them into the JDP. For more information on importing repositories, see [Import Repositories](#).

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
```

#### Note

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.

### 3.3.2 | System Import and Export

Artifactory maintains all security entities (users, groups, permissions and access tokens) when doing a system import.

#### Note

The System Import and Export feature is not available for Artifactory Cloud users.

#### Copy the \$JFROG\_HOME Master Key

Copying the `$JFROG_HOME/artifactory/var/etc/security/master.key` is a critical step in the import process - **for both single node and HA imports**. The `master.key` is an AES secret key (128 or 256 bit) that is used by Artifactory for encrypting and decrypting the shared data in the database and is, therefore, required for the import process.

#### Warning

We do not recommend that you import and export between different versions of Artifactory.

#### Tip

It is very important that the import files on the server should be readable and owned by the same user running the Artifactory process.

#### Note

Although we do not recommend it, when you import from a 6.x export into a 7.x installation, you need to replace the `logback.xml` in the export with a 7.x `logback.xml` otherwise logging will break during import.

To access import and export of your entire system, in the **Administration** module, select **Artifactory | Import & Export | System**.

Field	Description
Export Path on Server	The target directory for the exported files. You may browse your file system to select the directory
Exclude Content	<b>Export:</b> When set, repository binaries are excluded from the export. <b>Import:</b> When set, binaries and metadata are excluded from the import. Only builds and configuration files are imported.
Exclude Metadata	When set, repository metadata are excluded from the import/export. (Maven 2 metadata is unaffected by this setting) <b>Docker repositories must have metadata</b> For Docker repositories to work they must have their metadata intact. Therefore, if you have Docker repositories, make sure that <b>Exclude Metadata</b> is not checked when doing a system export or import.
Create .m2 Compatible Export	When set, includes Maven 2 repository metadata and checksum files as part of the export
Create a Zip Archive (Slow and CPU Intensive!)	When set, creates and exports to a zip archive. <b>Note</b> If there is failure when you import a zip file, extract the zip file to a folder and select the extracted folder for 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. <b>Monitoring the log</b> You can monitor the log in the System Logs page.

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:

**Note**

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.

**3.3.2.1 | 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.

**Note**

When importing from a 6.x export into a 7.x installation, you will need to replace the logback.xml in the export with a 7.x logback.xml, otherwise logging will break during import.

[Set a Target Artifactory](#)

1. Perform a normal system export from the source cluster as described above.
2. In the target cluster, keep only one node running, and perform a graceful shutdown to all the rest of the nodes.
3. Perform normal system import to the target cluster (which now has only one node running) as described above.
4. For Artifactory versions below 7.12.0, perform a graceful shutdown of the running node and then restart it.
5. For each additional node:
  - a. Delete the following folders:
    - \$JFROG\_HOME/artifactory/var/etc/access
    - \$JFROG\_HOME/artifactory/var/etc/security
    - \$JFROG\_HOME/artifactory/var/etc/artifactory/ui
    - \$JFROG\_HOME/artifactory/var/etc/artifactory/plugins
  - b. Copy the \$JFROG\_HOME/artifactory/var/etc/security/master.key from the running node to the additional nodes.
  - c. Start up the additional nodes.

Once you have completed the import, we recommend verifying that your HA cluster is up and running by checking whether all the nodes and services are online. For more information on viewing service status, see [Monitoring Service Statuses](#).

**Note**

After an import, the services of Distribution, Xray, Pipelines, and Insight should be restarted.

**3.3.2.2 | System Import and Active Repository Federations**

If you perform a system import on an Artifactory instance that has active repository Federations, all remote members are removed from the Federation automatically. This is done to ensure the import can be completed successfully.

After the import is complete, perform the procedure described below to recover the Federation.

**To recover a repository Federation after a system import:**

1. Re-establish trust with each remote member. For more information, see [Setup Prerequisites for Federated Repositories](#).
2. Go to the Artifactory instance (JPD) of each remote member, remove the imported member from the Federation, and then add it back again.

Adding the imported member back to the Federation will trigger a [Full Sync](#) for recovering the files.

### 3.4 | Backups

Artifactory enables you to configure the automatic backup of your system settings, then provide access to that backup in the case you need to restore it.

#### Note

Starting from Artifactory 7.66, the server path to a backup can no longer start with `$JFROG_HOME/artifactory`.

The following topics describe system backup and restore:

- [Complete System Backup](#)
- [Restoring a System Backup](#): To restore a system backup, you need to perform a system import. For more information, see [System Import and Export](#)

#### 3.4.1 | 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. Backup content is stored in standard file system format and can be loaded into any repository, so that Artifactory never locks you out.

#### Note

- [Backing up very large filestores](#): If you are backing up more than 1TB of storage, refer to this article for instructions.
- [Deleting outdated files to reduce backup storage](#): It is recommended to manually delete the `access.backup.<timestamp>.json` file created when performing backup or system export. This file can be found in the `$JFROG_HOME/artifactory/var/backup/access/` folder.

#### Prerequisites

In some user environments, the primary node is specifically configured with access to some NFS mount for Artifactory backups. With the introduction of Cloud-Native High Availability, where any node can create a backup, you need to set up access for all nodes to have *write* access to the mount for creating a backup. Alternatively, you can exclude all nodes from managing cluster-wide tasks except for a single node.

#### Path for Backup File

Starting from Artifactory version 7.66.x, the server path to backup can no longer start with `$JFROG_HOME/artifactory` directory. Therefore, you must store the backup outside the JFrog directory. The artifactory user must have access to the backup folder.

If you have any backups configured with the `$JFROG_HOME/artifactory` directory, you need to modify the path.

If you are moving to Cloud-Native High Availability on HA, we recommend that you have a shared drive path for backup paths (if you use a local drive path, the backup will be saved in whichever node triggers the backup operation, which can lead to confusion).

#### Note

To define multiple backups, in the **Administration** module, select **Artifactory | Backups**. Each backup may have its own schedule and repositories to either process or exclude.

#### To create a system backup:

1. In the navigation panel on the left-hand side of the screen, click **Administration** .
2. In the General section of this screen, click **Settings**. The **Artifactory General Settings** screen appears.
3. In the **Services** pane, click **Backups**. Backup settings are displayed.

# JFrog Artifactory Documentation

## Displayed in the header

4. From the top right-hand corner of the screen, click **New Backup**. The Backup Settings window is displayed. If previously configured, you can also choose an existing backup and change its properties.

5. Configure backup settings as required. When completed click **Save**. Settings are saved and the Backup is run when scheduled.

Field	Description
Enabled	When selected, the backup is enabled.
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: 0 0 /12 * * ?
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 Each run of this backup will create a new directory under this one with the timestamp as its name. <b>Path for Backup File</b> Starting from Artifactory version 7.66.x, the server path to backup can no longer start with \$JFROG_HOME/artifactory directory. Therefore, you must store the backup outside the JFrog directory. The artifactory user must have access to the backup folder. If you have any backups configured with the \$JFROG_HOME/artifactory directory, you need to modify the path.
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 New Repositories	To exclude new repositories, you'll need to add the <b>artifactory-build-info</b> repository to the Excluded Repositories.
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 <b>current</b> (as opposed to using the timestamp). The backup files can be used by any incremental file-system based backup utility (such as <b>rsync</b> ).
Retention Period Hours	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. <b>Warning</b> Do not store any custom files under the target backup directory, since the automatic backup cleanup processes may delete them!
Back up to a Zip Archive (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 \$JFROG\_HOME/artifactory/var/log/artifactory-service.log file. To monitor 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.
```

### 3.4.2 | Restoring a System Backup

To restore a system backup, you need to perform a system import. For more information, see [System Import and Export](#).

### 3.5 | Maven Indexer

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.

#### 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 Cloud?

The Maven Indexer service is only available on Artifactory Cloud **dedicated servers**.

#### Maven Indexer Usage

##### To configure the Maven Indexer:

1. In the **Administration** workspace, under **Artifactory Settings**, click **Maven Indexer**. Maven Indexer options are displayed.
2. In the **Services** pane, click **Maven Indexer**. Maven Indexer Settings are displayed.

3. Configure settings as shown in the following table. Artifactory provides you with controls to specify how frequently indexing is run and which repositories are included in the index calculation. When done, click Save. Your settings are saved.

Field	Description
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.
Excluded Repositories	Specifies the repositories that should not be indexed on the next run.
Included Repositories	Specifies the repositories that should be indexed on the next run.

Field	Description
Clear All	Removes all repositories from the <b>Included Repositories</b> 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.

### 3.6 | Artifactory Security

You can configure the following Artifactory security settings:

- [Artifactory Security - General Settings](#)
- [Artifactory Security - Certificates](#)

For information on how to configure additional Artifactory security settings, see the following:

- [Security Keys Management](#)
- [Managing Signing Keys](#)
- [Managing WebStart and Jar Signing](#)
- [Managing Public Keys](#)
- [Managing SSH Keys](#)

#### 3.6.1 | Artifactory Security - General Settings

Artifactory provides several system-wide settings to control access to different resources. Go to the [Administration module](#) and then go to [Artifactory | Security](#).

Field	Description
Allow Basic Read of Build Related Info	This setting, when enabled, gives all users (including anonymous users) view permissions to published modules for all builds in the system. This is regardless of any specific permissions applied to a particular build.
API Keys Management	The export target directory on your server. To revoke all API keys in the system, click <a href="#">Revoke API Keys for All Users</a> .

#### 3.6.2 | Artifactory Security - Certificates

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.

##### 3.6.2.1 | Add Certificates

Certificates are managed in the [Administration module](#) under [Artifactory | 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**.

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.

**Tip**

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.

3.6.2.2 | **Use 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.

3.6.2.3 | **Proxy a Resource that Uses a Self-Signed Certificates**

# JFrog Artifactory Documentation

## Displayed in the header

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, see Trust a Signed Certificate or a New CA.

### You cannot configure a self-signed certificate in Artifactory SaaS

If you are using Artifactory SaaS (as opposed to an self-hosted 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.

## 3.7 | Artifactory Configuration Descriptors

### Note

This page has been **deprecated**. For the most recent information, see Configuration Files.

All Artifactory configuration files are located under the `$JFROG_HOME/artifactory/var/etc/artifactory` folder.

On Linux, Solaris and MacOS `$JFROG_HOME` is usually a soft link to `opt/jfrog`.

### 3.7.1 | Global Configuration Descriptor

The global Artifactory configuration file is used to provide a default set of configuration parameters.

The file is located in `$JFROG_HOME/artifactory/var/etc/artifactory.config.latest.xml` and is loaded by Artifactory at initial startup.

Once the file is loaded, Artifactory renames it to `$JFROG_HOME/artifactory/var.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.

At any time, the default configuration can be changed in the Artifactory UI **Administration** module.

There are two ways to directly modify the Global Configuration Descriptor:

- Using the Artifactory UI
- Using the REST API

### Back Up Before Running Procedure

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.

### Note

From Artifactory version 7.49.x and forward, the Global Configuration Descriptor file will no longer contain Repository Configuration. You can use the Repositories REST API to modify repositories configuration. For more information, see Repositories REST API.

#### 3.7.1.1 | Modify Descriptor Configuration Using the UI

You can access the Configuration Descriptor in the **Administration** module under **Artifactory | Advanced | Config Descriptor**. There you can modify the file's contents directly or copy the contents from the entry field.

#### 3.7.1.2 | Modify Descriptor 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:

# JFrog Artifactory Documentation

## Displayed in the header

### 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
```

#### 3.7.1.3 | Bootstrap the Global Configuration

You can bootstrap Artifactory with a predefined global configuration by creating an `$JFROG_HOME/artifactory/var/etc/artifactory/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.

### 3.7.2 | Repository Descriptor Configuration

#### Note

Repositories Configuration is only available from version 7.49.x. For earlier versions, please use the Global Configuration Descriptor.

Artifactory periodically persists its current repositories configuration to `$JFROG_HOME/artifactory/var/etc/artifactory/artifactory.repository.config.latest.json`.

To modify the configuration of repositories, use the REST API endpoints or the repository configuration platform UI. For more information about the available options, see Repository Configuration JSON.

#### 3.7.3 | Security Configuration Descriptor

There are two ways to directly modify the Security Configuration Descriptor:

1. Using the Artifactory UI
2. Using the REST API

#### Back Up Before Running Procedure

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.

#### 3.7.3.1 | Modify Security Descriptor Using the UI

You can access the Security Configuration Descriptor in the **Administration** module under **Artifactory | Advanced | Security Descriptor**. There you can modify the file's contents directly or copy the contents from the entry field.

#### 3.7.3.2 | Modify Security Descriptor 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:

#### Modifying the Security Descriptor

```
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
```

# JFrog Artifactory Documentation

## Displayed in the header

### Admin privileges

You must supply a user with **Admin** privileges to modify the security descriptor through the REST API

#### 3.7.3.3 | Bootstrap 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 an `$JFROG_HOME/artifactory/var/etc/artifactory/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.

#### 3.7.4 | 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 `$JFROG_HOME/artifactory/var/etc/artifactory/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 those in the `mimetype.xml` file, and applies the MIME type of the first extension that matches.

##### Note

The ability to define and modify MIME types is available only to users working on a Self-Hosted platform.

#### 3.7.4.1 | MIME Type Attributes

Each MIME type may have the following attributes:

MIME Type Attribute	Description
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 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.

# JFrog Artifactory Documentation

## Displayed in the header

### 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.

#### 3.7.4.2 | Set Content-Type During Download

Using Artifactory, when downloading files you can override the `Content-Type` HTTPheader 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.

#### 3.7.5 | Descriptor Related System Properties

Rather than configuring properties in the JVM runtime configuration of the hosting container, you can edit `$JFROG_HOME/var/etc/artifactory/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.).

### Note

Setting properties in `artifactory.system.properties` is an advanced feature and is typically not required.

Do not confuse these setting with those in the `$JFROG_HOME/artifactory/var/etc/artifactory/artifactory.properties` file, which are for internal use.

## 3.8 | Artifactory Configuration XSD

All Artifactory configuration files can be found under the `$JFROG_HOME/artifactory/var/etc/artifactory` folder. Unlike the default `artifactory.config.xml`, which can be used as an example, but it does not contain all the fields), the Artifactory `artifactory.xsd` file contains all of the fields and descriptions available for Artifactory. In this document, you will find the latest version of the `artifactory.xsd` file and the available configuration options.

### Latest Artifactory XSD File

In this section, you will find the latest version of the `artifactory.xsd` file.

#### Artifactory XSD File

```
<?xml version="1.0" encoding="utf-8"?>
<!--
~ Artifactory is a binaries repository manager.
~ Copyright (C) 2018 JFrog Ltd.
~
~ Artifactory is free software: you can redistribute it and/or modify
~ it under the terms of the GNU Affero General Public License as published by
~ the Free Software Foundation, either version 3 of the License, or
~ (at your option) any later version.
~
~ Artifactory is distributed in the hope that it will be useful,
~ but WITHOUT ANY WARRANTY; without even the implied warranty of
~ MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
~ GNU Affero General Public License for more details.
~
~ You should have received a copy of the GNU Affero General Public License
~ along with Artifactory. If not, see <http://www.gnu.org/licenses/>.
-->

<xsschema xmlns:xss="http://www.w3.org/2001/XMLSchema" version="1.0"
    targetNamespace="http://artifactory.jfrog.org/xsd/3.3.9"
    xmlns="http://artifactory.jfrog.org/xsd/3.3.9"
    elementFormDefault="qualified">

    <xselement name="config" type="CentralConfigType"/>

    <xsccomplexType name="CentralConfigType">
        <xsssequence>
            <xselement name="repositoriesGeneralConfig" type="RepositoriesGeneralType" minOccurs="0">
                <xssannotation>
                    <xssdocumentation source="description">
                        Repositories-related global configuration.
                    </xssdocumentation>
                </xssannotation>
            </xselement>
            <xselement name="forceNativeDbLocksMechanismInAutoMode" type="xs:boolean" default="false" minOccurs="0">
                <xssannotation>
                    <xssdocumentation source="description">
                        If marked, in case Artifactory DB locks mechanism is in AUTO mode, it will switch to the Native
                        DB locks mechanism regardless of defined supported version
                    </xssdocumentation>
                </xssannotation>
            </xselement>
        </xsssequence>
    </xsccomplexType>
</xsschema>
```

# JFrog Artifactory Documentation Displayed in the header

```
<xss:element name="useOnlyTokenForIdentifyingXrayUser" type="xs:boolean" default="true" minOccurs="0">
    <xss:annotation>
        <xss:documentation source="description">
            If set to true Artifactory will use only an authentication token for identifying XRay users
            ignoring the user-agent header and other authentication types. It is used for building
            S3 redirect URLs only.
        </xss:documentation>
    </xss:annotation>
</xss:element>
<xss:element name="serverName" type="xs:string" minOccurs="0">
    <xss:annotation>
        <xss:documentation source="description">
            <![CDATA[
                A name uniquely identifying this artifactory server instance across the network.
            ]]>
        </xss:documentation>
    </xss:annotation>
</xss:element>
<xss:element name="offlineMode" type="xs:boolean" default="false" minOccurs="0">
    <xss:annotation>
        <xss:documentation source="description">
            If marked, Artifactory never tries to fetch artifacts remotely.
            Only cached and local artifacts are served.
        </xss:documentation>
    </xss:annotation>
</xss:element>
<xss:element name="archiveIndexEnabled" type="xs:boolean" default="false" minOccurs="0">
    <xss:annotation>
        <xss:documentation source="description">
            If marked, Artifactory will index the archive content and enable search within the archive.
        </xss:documentation>
    </xss:annotation>
</xss:element>
<xss:element name="fileUploadMaxSizeMb" type="xs:int" default="100" minOccurs="0">
    <xss:annotation>
        <xss:documentation source="description">
            The maximum size in megabytes for artifact files uploaded through the web UI. Set to '0' for
            unlimited size.
        </xss:documentation>
    </xss:annotation>
</xss:element>
<xss:element name="revision" type="xs:long" default="0" minOccurs="0">
    <xss:annotation>
        <xss:documentation source="description">
            Do not update and do not delete. The revision of a configuration file to compare, managed internal.
        </xss:documentation>
    </xss:annotation>
</xss:element>
<xss:element name="addons" type="AddonsType" minOccurs="0">
    <xss:annotation>
        <xss:documentation source="description">
            Add-ons-related configuration.
        </xss:documentation>
    </xss:annotation>
</xss:element>
<xss:element name="mailServer" type="MailServerType" minOccurs="0">
    <xss:annotation>
        <xss:documentation source="description">
            Mail server-related configuration.
        </xss:documentation>
    </xss:annotation>
</xss:element>
<xss:element name="xrayConfig" type="XrayType" minOccurs="0">
    <xss:annotation>
        <xss:documentation source="description">
            Xray related configuration.
        </xss:documentation>
    </xss:annotation>
</xss:element>
<xss:element name="security" type="SecurityType" minOccurs="0">
    <xss:annotation>
        <xss:documentation source="description">
            Security-related configuration used in authentication and authorization.
        </xss:documentation>
    </xss:annotation>
</xss:element>
<xss:element name="backups" type="BackupsType" minOccurs="0">
    <xss:annotation>
        <xss:documentation source="description">
            A set of backup configurations.
        </xss:documentation>
    </xss:annotation>
</xss:element>
<xss:element name="indexer" type="IndexerType" minOccurs="0"/>
<xss:element name="localRepositories" type="LocalRepositoriesType" minOccurs="0">
    <xss:annotation>
        <xss:documentation source="description">
            A set of locally installed repositories.
        </xss:documentation>
    </xss:annotation>
</xss:element>
<xss:element name="remoteRepositories" type="RemoteRepositoriesType" minOccurs="0">
    <xss:annotation>
        <xss:documentation source="description">
```

# JFrog Artifactory Documentation Displayed in the header

```
A set of remote proxied (normally cached) repositories.  
</xs:documentation>  
</xs:annotation>  
</xs:element>  
<xs:element name="virtualRepositories" type="VirtualRepositoriesType" minOccurs="0">  
<xs:annotation>  
<xs:documentation source="description">  
A set of virtual repositories aggregating regular local and remote repositories.  
</xs:documentation>  
</xs:annotation>  
</xs:element>  
<xs:element name="federatedRepositories" type="FederatedRepositoriesType" minOccurs="0">  
<xs:annotation>  
<xs:documentation source="description">  
A set of federated repositories which are used for complete multi directional sync between  
repos.  
</xs:documentation>  
</xs:annotation>  
</xs:element>  
<xs:element name="releaseBundlesRepositories" type="ReleaseBundlesRepositoriesType" minOccurs="0">  
<xs:annotation>  
<xs:documentation source="description">  
A set of release bundles repositories which are used to distribute release bundles between  
distribution and artifactory  
</xs:documentation>  
</xs:annotation>  
</xs:element>  
<xs:element name="proxies" minOccurs="0">  
<xs:annotation>  
<xs:documentation source="description">  
A set of reusable proxy definitions. Proxies can be used by remote repositories by associating a  
proxy definition with a remote repository.  
</xs:documentation>  
</xs:annotation>  
<xs:complexType>  
<xs:sequence>  
<xs:element name="proxy" type="ProxyType" maxOccurs="unbounded" minOccurs="0"/>  
</xs:sequence>  
</xs:complexType>  
</xs:element>  
<xs:element name="reverseProxies" minOccurs="0">  
<xs:annotation>  
<xs:documentation source="description">  
A set of reusable reverse proxy definitions.  
</xs:documentation>  
</xs:annotation>  
<xs:complexType>  
<xs:sequence>  
<xs:element name="reverseProxy" type="ReverseProxyType" maxOccurs="unbounded" minOccurs="0"/>  
</xs:sequence>  
</xs:complexType>  
</xs:element>  
<xs:element name="propertySets" type="PropertySetsType" minOccurs="0">  
<xs:annotation>  
<xs:documentation source="description">  
Define property sets to group together properties of the same category (e.g. QA). Then, define  
properties and their respective possible values (e.g. performance:{poor,acceptable,good} and  
qaStaffComment:[any]).  
Once property sets and their properties are defined, you can associate them with repositories,  
in order to be able to easily set, update and search for predefined property values on hosted  
repository items.  
</xs:documentation>  
</xs:annotation>  
</xs:element>  
<xs:element name="systemProperties" type="SystemPropertiesType" minOccurs="0">  
<xs:annotation>  
<xs:documentation source="description">  
A set of key:value reloadable system properties  
</xs:documentation>  
</xs:annotation>  
</xs:element>  
<xs:element name="urlBase" type="xs:string" default="" minOccurs="0">  
<xs:annotation>  
<xs:documentation source="description">  
A hard-coded URL prefix used by Artifactory for calculating relative URLs.  
Use this only if you must override the default URL base (similar to the Servlet context URL  
- e.g. http://myhost.com/artifactory).  
</xs:documentation>  
</xs:annotation>  
</xs:element>  
<xs:element name="federatedRepoUrlBase" type="xs:string" default="" minOccurs="0" maxOccurs="1">  
<xs:annotation>  
<xs:documentation source="description">  
A hard-coded URL prefix used by federated repository as the local member URL base.  
federatedRepoUrlBase is overriding the regular urlBase  
- e.g. http://myhost.com/artifactory).  
</xs:documentation>  
</xs:annotation>  
</xs:element>  
<xs:element name="footer" type="xs:string" default="" minOccurs="0">  
<xs:annotation>  
<xs:documentation source="description">  
Additional custom text to appear in the page footer.  
</xs:documentation>
```

# JFrog Artifactory Documentation Displayed in the header

```
</xs:annotation>
</xs:element>
<xs:element name="repoLayouts" minOccurs="1">
    <xs:annotation>
        <xs:documentation source="description">
            Define reusable repository storage layouts used for identifying modules (module artifacts and module descriptors).
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="repoLayout" type="RepoLayoutType" maxOccurs="unbounded" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="remoteReplications" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            Configure scheduled remote repository replications.
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="remoteReplication" type="RemoteReplicationType" maxOccurs="unbounded" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="localReplications" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            Configure scheduled local repository replications.
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="localReplication" type="LocalReplicationType" maxOccurs="unbounded" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="gcConfig" type="GcConfigType" minOccurs="1"/>
<xs:element name="cleanupConfig" type="CleanupConfigType" minOccurs="1"/>
<xs:element name="virtualCacheCleanupConfig" type="CleanupConfigType" minOccurs="1"/>
<xs:element name="quotaConfig" type="QuotaConfigType" minOccurs="0"/>
<xs:element name="folderDownloadConfig" type="FolderDownloadConfigType" minOccurs="1" maxOccurs="1"/>
<xs:element name="trashcanConfig" type="TrashcanConfigType" minOccurs="1" maxOccurs="1"/>
<xs:element name="itemsBrowserPageSize" type="xs:int" default="500" minOccurs="0" maxOccurs="1">
    <xs:annotation>
        <xs:documentation source="description">
            The artifacts paging size.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="replicationsConfig" type="ReplicationsConfigType" minOccurs="0" maxOccurs="1"/>
<xs:element name="sumologicConfig" type="SumologicConfigType" minOccurs="0" maxOccurs="1"/>
<xs:element name="releaseBundlesConfig" type="ReleaseBundlesConfigType" minOccurs="0" maxOccurs="1"/>
<xs:element name="signedUrlConfig" type="SignedUrlConfigType" minOccurs="0" maxOccurs="1"/>
<xs:element name="downloadRedirectConfig" type="DownloadRedirectConfigType" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            Default (globally used) download redirect configuration.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="eula" type="EulaType" minOccurs="0" maxOccurs="1">
    <xs:annotation>
        <xs:documentation source="description">
            Eula agreement.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="subscription" type="Subscription" minOccurs="0" maxOccurs="1">
    <xs:annotation>
        <xs:documentation source="description">
            JCR Newsletter Subscription.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="keyPairs" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            A set of reusable key pairs definitions. Key pairs can be used by local/virtual repositories by associating a keypair definition with them, in order to sign metadata files.
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="keyPair" type="KeyPairType" maxOccurs="unbounded" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="retentionSettings" minOccurs="0">
    <xs:complexType>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
<xs:sequence>
  <xs:element name="jfrogColdStorageArchivingNamespace" type="xs:string" minOccurs="0">
    <xs:annotation>
      <xs:documentation source="description">
        The Archiving Namespace identifier on the JFrog Cold Storage instance
      </xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="jfrogColdStorageMasterToken" minOccurs="0" type="RetentionMasterTokenType">
  </xs:element>
  <xs:element name="retentionPolicies" minOccurs="0">
    <xs:annotation>
      <xs:documentation source="description">
        A set of reusable retention policies. Policies can be triggered to execute
        manual or scheduled artifact retention tasks.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="retentionPolicy" type="RetentionPolicyType"
          maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  </xs:sequence>
</xs:complexType>
</xs:complexType>
<xs:element name="authentication" type="AuthenticationType" minOccurs="0"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="LocalRepositoriesType">
  <xs:sequence>
    <xs:element name="localRepository" type="LocalRepoType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="LocalRepoType">
  <xs:complexContent>
    <xs:extension base="RealRepoType">
      <xs:sequence>
        <xs:element name="snapshotVersionBehavior" default="unique" minOccurs="0">
          <xs:annotation>
            <xs:documentation source="description">
              Whether to use time-based version numbers for the name of SNAPSHOTs,
              or to use a non-unique, self-overriding naming pattern of
              artifactId-version-SNAPSHOT.type (default), or to respect the
              deployer settings coming from the Maven client.
            </xs:documentation>
          </xs:annotation>
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="deployer"/>
              <xs:enumeration value="non-unique"/>
              <xs:enumeration value="unique"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="localRepoChecksumPolicyType" default="client-checksums" minOccurs="0">
          <xs:annotation>
            <xs:documentation source="description">
              Checksum policy determines how Artifactory behaves when a client checksum for a deployed
              resource is missing or conflicts with the locally calculated checksum (bad checksum).
              Checking checksums effectively verifies the integrity of the deployed resource.
            </xs:documentation>
            The options are:
            (1) Verify against client checksums (default) - Until a client has sent a valid checksum
            for a deployed artifact matching the server's locally calculated checksum, the
            artifact's checksum is not available and returns 404 (not found).
            If the client has sent a checksum conflicting with the one calculated on the server a
            409 (conflict) is returned until a valid checksum is deployed.
            (2) Trust server generated checksums - Do not verify against checksums sent by clients
            and trust the store server's locally calculated checksums. An uploaded artifact is
            available for consumption immediately, but integrity might be compromised.
          </xs:annotation>
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="client-checksums"/>
              <xs:enumeration value="server-generated-checksums"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="calculateYumMetadata" type="xs:boolean" default="false" minOccurs="0">
          <xs:annotation>
            <xs:documentation source="description">
              A-synchronously calculate YUM repository metadata upon RPM deployment and removal.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="yumRootDepth" type="xs:int" default="0" minOccurs="0">
          <xs:annotation>
            <xs:documentation source="description">
              The depth, relative to the repository's root folder, where YUM metadata is created.
              This is useful when your repository contains multiple YUM repositories under parallel
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
 hierarchies.  
 For example, if your RPMs are stored under 'fedora/linux/$releasever/$basearch', specify  
 For example, if your RPMs are stored under 'fedora/linux/$releasever/$basearch', specify  
 a depth of 4.  
 </xs:documentation>  
 </xs:annotation>  
</xs:element>  
<xs:element name="yumGroupFileNames" type="xs:string" minOccurs="0">  
 <xs:annotation>  
   <xs:documentation source="description">  
     A comma-separated list of xml file names containing YUM group component definitions.  
     Artifactory includes the group definitions as part of the calculated YUM metadata,  
     as well as automatically generating a zipped version of the group files, if required.  
   </xs:documentation>  
 </xs:annotation>  
</xs:element>  
<xs:element name="debianTrivialLayout" type="xs:boolean" default="true" minOccurs="0">  
 <xs:annotation>  
   <xs:documentation source="description">  
     The default Debian architecture is "Automatic", to change the Debian architecture to  
     'Trivial' (which is deprecated) and generate the indices in the root directory,  
     set the Trivial Layout checkbox.  
   </xs:documentation>  
 </xs:annotation>  
</xs:element>  
<xs:element name="enableFileListsIndexing" type="xs:boolean" default="false" minOccurs="0">  
 <xs:annotation>  
   <xs:documentation source="description">  
     When true, Artifactory will index the filelists.xml metadata file which contains a list  
     of all files in every RPM package in the repository.  
   </xs:documentation>  
 </xs:annotation>  
</xs:element>  
<xs:element name="optionalIndexCompressionFormats" type="FormatType" minOccurs="0">  
 <xs:annotation>  
   <xs:documentation source="description">  
     List of optional index compression formats for debian repository.  
   </xs:documentation>  
 </xs:annotation>  
</xs:element>  
<xs:element name="dockerTagRetention" type="xs:int" default="1" minOccurs="0" maxOccurs="1">  
 <xs:annotation>  
   <xs:documentation source="description">  
     Max number of Docker images with the same tag.  
   </xs:documentation>  
 </xs:annotation>  
</xs:element>  
<xs:element name="enableComposerV1Indexing" type="xs:boolean" default="false" minOccurs="0">  
 <xs:annotation>  
   <xs:documentation source="description">  
     When true, Artifactory will create Composer metadata v1 index files alongside the  
     metadata v2 index files.  
   </xs:documentation>  
 </xs:annotation>  
</xs:element>  
<xs:element name="terraformType" type="xs:string" minOccurs="0">  
 <xs:annotation>  
   <xs:documentation source="description">  
     Terraform local repository type between the following:  
     1. module  
     2. provider  
   </xs:documentation>  
 </xs:annotation>  
</xs:element>  
</xs:sequence>  
</xs:extension>  
</xs:complexContent>  
</xs:complexType>  
  
<xs:complexType name="FederatedRepositoriesType">  
 <xs:sequence>  
   <xs:element name="federatedRepository" type="FederatedRepoType" minOccurs="0" maxOccurs="unbounded"/>  
 </xs:sequence>  
</xs:complexType>  
  
<xs:complexType name="ReleaseBundlesRepositoriesType">  
 <xs:sequence>  
   <xs:element name="releaseBundlesRepository" type="ReleaseBundlesRepoType" minOccurs="0"  
   maxOccurs="unbounded"/>  
 </xs:sequence>  
</xs:complexType>  
  
<xs:complexType name="ReleaseBundlesRepoType">  
 <xs:complexContent>  
   <xs:extension base="LocalRepoType">  
     </xs:extension>  
   </xs:complexContent>  
</xs:complexType>  
  
<xs:complexType name="FederatedRepoType">  
 <xs:complexContent>  
   <xs:extension base="LocalRepoType">  
     <xs:sequence>  
       <xs:element name="federatedMembers" type="FederatedMembersType" minOccurs="0">  
         </xs:element>  
     </xs:sequence>  
   </xs:extension>  
</xs:complexContent>  
</xs:complexType>
```

# JFrog Artifactory Documentation Displayed in the header

```
<xs:annotation>
    <xs:documentation source="description">
        List of distribution rules used by this repository.
    </xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="proxyRef" type="xs:IDREF" minOccurs="0" maxOccurs="1">
    <xs:annotation>
        <xs:documentation source="description">
            Network proxy reference.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="disableProxy" type="xs:boolean" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            Turns on/off proxy usage for this repository.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="modificationDate" type="xs:long" minOccurs="0" maxOccurs="1">
    <xs:annotation>
        <xs:documentation source="description">
            Configuration last modificationDate
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="maxNumberOfThreads" type="xs:integer" minOccurs="0" maxOccurs="1">
    <xs:annotation>
        <xs:documentation source="description">
            The max number of threads handling async operations triggered by federation events
        </xs:documentation>
    </xs:annotation>
</xs:element>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<xs:complexType name="RealRepoType" abstract="true">
    <xs:complexContent>
        <xs:extension base="RepoType">
            <xs:sequence>
                <xs:element name="blacklisted" type="xs:boolean" default="false" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            Whether the repository is blacked-out. A blacked-out repository or its local cache do
                            not
                            participate in artifact resolution.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="handleReleases" type="xs:boolean" default="true" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            Whether the repository handles release artifacts.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="handleSnapshots" type="xs:boolean" default="true" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            Whether the repository handles snapshot artifacts.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="maxUniqueSnapshots" type="xs:int" default="0" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            The maximum number of unique snapshots (of the same artifact) to store.
                            Any number of snapshots above the max are automatically removed by age.
                            A value of 0 (the default) indicates no limits on unique snapshots number (thus
                            no automatic cleanup).
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="maxUniqueTags" type="xs:int" default="0" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            The maximum number of unique tags (of the same docker native repository) to store.
                            Any number of tags above the max are automatically removed by age.
                            A value of 0 (the default) indicates no limits on unique snapshots number (thus
                            no automatic cleanup).
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="blockPushingSchema1" type="xs:boolean" default="true" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            When set, Artifactory will block the pushing of Docker images with manifest v2 schema 1 to this repository.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="suppressPomConsistencyChecks" type="xs:boolean" default="true" minOccurs="0">
```

# JFrog Artifactory Documentation

## Displayed in the header

```
<xs:annotation>
    <xs:documentation source="description">
        Whether the repository should suppress POM consistency checks.
    </xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="propertySets" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            The property sets that you are able to use of items hosted under this repository.
            As property sets are purely assistive and not enforcing, setting this on a repository
            only affects the list of predefined properties in the user interface properties tab for
            items under the repository. You are still be able to attach arbitrary properties to
            repository items.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:complexType>
    <xs:sequence>
        <xs:element name="propertySetRef" type="xs:IDREF" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
                <xs:documentation source="description">
                    A property set usable by this repository.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="archiveBrowsingEnabled" type="xs:boolean" minOccurs="0" default="false">
    <xs:annotation>
        <xs:documentation source="description">
            <![CDATA[
                Allow Unsafe Content and Archive Browsing -
                Determines whether viewing content (e.g., html files, Javadoc browsing) directly from Artifactory is allowed.
                Allowing content browsing requires strict content moderation in order to make sure malicious users do not
                upload content that is used to compromise security. For example, to conduct cross-site scripting attacks.
            ]]>
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="xray" type="RepoXrayConfigType" minOccurs="0" maxOccurs="1">
    <xs:annotation>
        <xs:documentation source="description">
            Xray configuration for this repository.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="downloadRedirect" type="RepoDownloadRedirectConfigType" minOccurs="0" maxOccurs="1">
    <xs:annotation>
        <xs:documentation source="description">
            Download redirect configuration for this repository. When set TRUE: all download requests
            to this repo will be answered with a HTTP 302 response with "location" header.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="priorityResolution" type="xs:boolean" minOccurs="0" default="false">
    <xs:annotation>
        <xs:documentation source="description">
            Priority resolution means on supported packages packages that exists both on priority
            and non priority resolution repositories will only have their metadata merged from the
            priority ones, based on each package logic.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="cdnRedirect" type="RepoCdnRedirectConfigType" minOccurs="0" maxOccurs="1">
    <xs:annotation>
        <xs:documentation source="description">
            CDN redirect configuration for this repository. When set TRUE: all download requests to
            this repo will be answered with a HTTP 302 to response with "location" header.
        </xs:documentation>
    </xs:annotation>
</xs:element>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<xs:complexType name="RepoType" abstract="true">
    <xs:sequence>
        <xs:element name="key" type="xs:ID">
            <xs:annotation>
                <xs:documentation source="description">
                    Repository unique ID.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="type" type="xs:string" minOccurs="1"/>
        <xs:element name="description" type="xs:string" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    Textual description of the repository.
                    This description also appears when selecting the repository in the Tree Browser.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
</xs:element>
<xs:element name="notes" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      Optional notes about this repository.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="includesPattern" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      Comma-separated list of artifact patterns to include when evaluating
      artifact requests, in the form of x/y/**/z/*. When used, only requests
      matching one of the include patterns are served.
      By default all artifacts are included (**/*).
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="excludesPattern" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      Comma-separated list of artifact patterns to exclude when evaluating
      artifact requests, in the form of x/y/**/z/*. By default no artifacts are excluded.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="repoLayoutRef" type="xs:IDREF" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      Select the layout that the repository should use for storing and identifying modules.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="dockerApiVersion" default="V1" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation source="description">
      Docker API version.
    </xs:documentation>
  </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="V1"/>
      <xs:enumeration value="V2"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="forceNuGetAuthentication" type="xs:boolean" default="false" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      Force basic authentication credentials in order to use this repository.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="forceConanAuthentication" type="xs:boolean" default="false" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      Force basic authentication credentials in order to use this repository.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="ddebSupported" type="xs:boolean" default="false" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      Debian repo ".ddeb" packages support
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="cargoConfig" type="CargoConfigType" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation source="description">
      Cargo client specific authorization override
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="primaryKeyPairRef" type="xs:IDREF" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation source="description">
      Primary Key pair reference.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="secondaryKeyPairRef" type="xs:IDREF" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation source="description">
      Secondary Key pair reference.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="signedUrlTtl" type="xs:long" default="90" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      The number of seconds ttl for sign url.
    </xs:documentation>
  </xs:annotation>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="RemoteRepositoriesType">
<xs:sequence>
<xs:element name="remoteRepository" type="RemoteRepoType"
    minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="RemoteRepoType">
<xs:complexContent>
<xs:extension base="RemoteRepoBaseType">
<xs:sequence>
<xs:element name="username" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
HTTP authentication username.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="password" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
HTTP authentication password.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="allowAnyHostAuth" type="xs:boolean" default="false" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
Allow credentials of this repository to be used on requests redirected to any other
host.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="socketTimeoutMillis" type="xs:int" default="15000"
    minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
Network timeout in milliseconds to use both for connection
establishment and for unanswered requests.
Timing out on a network operation is considered as a retrieval
failure.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="enableCookieManagement" type="xs:boolean" default="false" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
Enable cookie management if the remote repo uses cookies to manage client state.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="enableTokenAuthentication" type="xs:boolean" default="false" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
Enable token (Bearer) based authentication
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="localAddress" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
The local address to be used when creating connections.
Useful for specifying the interface to use on systems with multiple network interfaces.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="proxyRef" type="xs:IDREF" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
Network proxy reference.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="disableProxy" type="xs:boolean" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
Turns on/off proxy usage for this repository.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="queryParams" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
<![CDATA[
Custom HTTP query parameters that will be automatically included in all remote resource requests.
For example: param1=val1&param2=val2&param3=val3
]]>
</xs:documentation>
</xs:annotation>
```

# JFrog Artifactory Documentation Displayed in the header

```
</xs:element>
<xs:element name="propagateQueryParams" type="xs:boolean" default="false" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            <![CDATA[
                If original query parameters should be included in remote requests
            ]]>
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="clientTlsCertificate" type="xs:string" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            The client certificate name
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="disableUrlNormalization" type="xs:boolean" default="false" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            Whether to disable URL normalization that is made by a HTTP client when accessing remote hosts.
            Should be used if, for example, a remote host redirects to a URL that contains double slash symbols and that is an expected behavior and "://" should not be changed to "/".
        </xs:documentation>
    </xs:annotation>
</xs:element>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<xs:complexType name="RemoteRepoBaseType" abstract="true">
    <xs:complexContent>
        <xs:extension base="RealRepoType">
            <xs:sequence>
                <xs:element name="url" type="xs:string">
                    <xs:annotation>
                        <xs:documentation source="description">
                            The URL for the remote repository. Currently only HTTP/S URLs are supported.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="offline" type="xs:boolean" default="false" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            When marked, only already-cached artifacts are retrieved and fetching remote artifacts is not attempted.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="hardFail" type="xs:boolean" default="false" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            Whether failing to communicate with this repository returns an error to the client that fails the build.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="storeArtifactsLocally" type="xs:boolean" default="true" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            Whether the repository should store cached artifacts locally or not. When not storing artifacts locally, direct repository-to-client streaming is used. This can be useful for multi-server setups over a high-speed LAN, with one Artifactory caching certain data on a central storage and streaming it directly to satellite pass-through Artifactory servers.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="fetchJarsEagerly" type="xs:boolean" default="true" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            When marked, the repository attempts to eagerly fetch the jar in the background each time a POM is requested.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="fetchSourcesEagerly" type="xs:boolean" default="true" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            When marked, the repository attempts to eagerly fetch the source jar in the background each time a jar is requested.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="externalDependencies" type="ExternalDependenciesConfigType" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            Configuration for fetching external dependencies
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
<xs:element name="retrievalCachePeriodSecs" type="xs:long" default="7200" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      The number of seconds to cache artifact look-up results.
      A value of 0 indicates no caching.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="metadataRetrievalTimeoutSecs" type="xs:long" default="60" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      The number of seconds to wait for lock to be acquired upon metadata download.
      If lock not acquire, the cached metadata is returned.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="assumedOfflinePeriodSecs" type="xs:long" default="30" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      The number of seconds the repository stays in assumed offline state after a connection
      error. At the end of this time an online check is attempted in order to reset the
      offline status.
      A value of 0 means the repository is never assumed offline.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="missedRetrievalCachePeriodSecs" type="xs:long" default="1800" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      The number of seconds to cache artifact retrieval misses (artifact not found).
      A value of 0 indicates no caching.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="remoteRepoChecksumPolicyType" default="generate-if-absent" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      Checksum policy determines how Artifactory behaves when a checksum for a remote resource
      is missing or conflicts with the locally calculated checksum (bad checksum).
      Checksum checking effectively verifies the integrity of the remote resource.
      The options are:
      1) Generate if absent (by default) - A bad remote checksum fails the resource
      request. If, however, a remote checksum could not be found Artifactory automatically
      generates one.
      2) Fail: A bad or missing remote checksum fails the resource request.
      3) Ignore and generate: Artifactory locally generates a checksum for both bad or
      missing remote checksum. Remote resource retrieval never fails, but integrity might
      be compromised.
      4) Pass-thru: Artifactory stores and passes-thorugh all remote checksums including
      bad ones and locally generates a checksum for a missing remote checksum that cannot be
      found. Remote resource retrieval never fails, but integrity might be compromised and
      client-side checksum validation (such as the one performed by Maven) fails.
    </xs:documentation>
  </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="generate-if-absent"/>
      <xs:enumeration value="fail"/>
      <xs:enumeration value="ignore-and-generate"/>
      <xs:enumeration value="pass-thru"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="unusedArtifactsCleanupPeriodHours" type="xs:int" default="0" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      The number of hours to wait before an artifact is deemed "unused" and eligible for
      cleanup from the repository.
      A value of 0 means automatic cleanup of cached artifacts is disabled.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="shareConfiguration" type="xs:boolean" default="false" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      Whether or not the configuration details of this remote repository can be publicly
      shared with remote clients, such as other Artifactory servers.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="synchronizeProperties" type="xs:boolean" default="false" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      Whether to synchronize properties of artifacts retrieved from a remote instance of
      Artifactory.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="listRemoteFolderItems" type="xs:boolean" default="true" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      Lists the items of remote folders in simple and list browsing. Required for dynamic
      resolution that depends on remote folder content information, such as remote Ivy version
      lookups. The remote content is cached according to the value of the
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

# JFrog Artifactory Documentation Displayed in the header

```
'Retrieval Cache Period'
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="remoteRepoLayoutRef" type="xs:IDREF" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
Select the layout that best matches the layout used by remote repository for storing and
identifying modules.
Path-mapping takes place if the remote layout is different from the local layout -
remote module artifacts and descriptors is stored according to the local repository
layout (e.g., Maven 1->Maven 2, or Maven 2->Ivy).
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="rejectInvalidJars" type="xs:boolean" default="false" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
Reject the caching of jar files that are found to be invalid. For example, pseudo jars
retrieved behind a "captive portal".
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="p2Support" type="xs:boolean" default="false" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
Support remote P2 metadata updates.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="nuget" type="NuGetConfigurationType" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
Proxy and cache NuGet libraries from the remote repository.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="pypi" type="PypiConfigurationType" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
Proxy and cache PyPI libraries from the remote repository.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="bower" type="BowerConfigurationType" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
Proxy and cache Bower libraries from the remote repository.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="cocoaPods" type="CocoaPodsConfigurationType" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
Proxy and cache CocoaPods libraries from the remote repository.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="helm" type="HelmConfigurationType" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
Proxy and cache Helm libraries from the remote repository.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="git" type="GitConfigurationType" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
A git registry of the repository
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="p2OriginalUrl" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
The URL for the original P2 repository. Currently only HTTP/S URLs are
supported.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="vcs" type="VcsType" minOccurs="0"/>
<xs:element name="contentSynchronisation" type="ContentSynchronisation" minOccurs="0" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
Content synchronisation to another artifactory repository
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="blockMismatchingMimeTypes" type="xs:boolean" default="true" minOccurs="0" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
If set, artifacts will fail to download if a mismatch is detected between requested and
received mimetype according to the list specified in the system properties file under
blockedMismatchingMimeTypes. You can override by adding mimetypes to the override list below.
</xs:documentation>
</xs:annotation>
</xs:element>
```

# JFrog Artifactory Documentation Displayed in the header

```
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="mismatchingMimeTypesOverrideList" type="xs:string" minOccurs="0" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
A comma separated list of mime types that will be blocked by this repo if the original
request was for a different mime type.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="bypassHeadRequests" type="xs:boolean" minOccurs="0" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
Before caching an artifact, Artifactory first sends a HEAD request to the remote resource. In some remote resources,
HEAD requests are disallowed and therefore rejected, even though downloading the artifact is allowed. When checked,
Artifactory will bypass the HEAD request and cache the artifact directly using a GET request.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="composer" type="ComposerConfigurationType" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
Proxy and cache Bower libraries from the remote repository.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="terraform" type="TerraformConfigurationType" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
Proxy and cache Terraform libraries from the remote repository.
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<xs:complexType name="ContentSynchronisation">
<xs:sequence>
<xs:element name="enabled" type="xs:boolean" form="qualified" default="false" minOccurs="1" maxOccurs="1"/>
<xs:element name="statistics" type="StatisticsContent" form="qualified" minOccurs="1" maxOccurs="1"/>
<xs:element name="properties" type="PropertiesContent" form="qualified" minOccurs="1" maxOccurs="1"/>
<xs:element name="source" type="SourceContent" form="qualified" minOccurs="1" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="StatisticsContent">
<xs:complexContent>
<xs:extension base="AbstractRemoteRepoDelegate">
<xs:sequence/>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<xs:complexType name="PropertiesContent">
<xs:complexContent>
<xs:extension base="AbstractRemoteRepoDelegate">
<xs:sequence/>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<xs:complexType name="SourceContent">
<xs:sequence>
<xs:element name="originAbsenceDetection" type="xs:boolean" form="qualified" default="false" minOccurs="1" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="AbstractRemoteRepoDelegate" abstract="true">
<xs:sequence>
<xs:element name="enabled" type="xs:boolean" form="qualified" default="false" minOccurs="1" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="VirtualRepositoriesType">
<xs:sequence>
<xs:element name="virtualRepository" type="VirtualRepoType" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="VirtualRepoType">
<xs:complexContent>
<xs:extension base="RepoType">
<xs:sequence>
<xs:element name="hideUnauthorizedResources" type="xs:boolean" minOccurs="0" default="false">
<xs:annotation>
<xs:documentation source="description">
When marked, Artifactory exposes the absence of unauthorized but nonexistent resources
by
sending a 404 response (not found) to requests for resources that are not accessible by
the user. Otherwise, the response implies that the resource exists but is protected -
by requesting authentication for anonymous requests (401) or by denying an authenticated
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
request for unauthorized users (403).
If hideUnauthorizedResources is marked in the security level, the virtual repo
level will has no effect.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="artifactoryRequestsCanRetrieveRemoteArtifacts" type="xs:boolean" minOccurs="0"
           default="false">
    <xs:annotation>
        <xs:documentation source="description">
            Determines whether artifact requests coming from other Artifactories
            can be fulfilled by accessing this virtual repository's remote repositories
            or by only accessing its caches (by default).
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="resolveDockerTagsByTimestamp" type="xs:boolean" minOccurs="0" default="false">
    <xs:annotation>
        <xs:documentation source="description">
            When enabled, in cases where the same Docker tag exists in two or more of the aggregated repositories,
            Artifactory will return the tag that has the latest timestamp.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="repositories" type="RepositoryRefsType" maxOccurs="1" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            A set of local and remote repository references to include in a
            virtual repository.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="keyPair" type="xs:string" minOccurs="0" maxOccurs="1">
    <xs:annotation>
        <xs:documentation source="description">
            A named key-pair to use for automatically signing artifacts.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="pomRepositoryReferencesCleanupPolicy" default="discard_active_reference"
           minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            (1) Discard Active References - Removes repository elements that are declared
                directly under project or under a profile in the same POM that is activeByDefault.
            (2)Discard Any References - Removes all repository elements regardless of whether
                they are included in an active profile or not.
            (3)Nothing - Does not remove any repository elements declared in the POM.
        </xs:documentation>
    </xs:annotation>
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="discard_active_reference"/>
            <xs:enumeration value="discard_any_reference"/>
            <xs:enumeration value="nothing"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
<xs:element name="p2" type="P2ConfigurationType" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            Configuration for automatic P2 handling
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="defaultDeploymentRepo" type="xs:IDREF" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            Local repository reference (key).
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="cachingLocalForeignLayersEnabled" type="xs:boolean" minOccurs="0" default="false">
    <xs:annotation>
        <xs:documentation source="description">
            Should use remote repository for caching docker foreign layers.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="externalDependencies" type="ExternalDependenciesConfigType" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            Configuration for fetching external dependencies
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="virtualCacheConfig" type="VirtualCacheConfigType" minOccurs="0" maxOccurs="1">
    <xs:annotation>
        <xs:documentation source="description">
            Configuration for yum virtual repo
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="forceMavenAuthentication" type="xs:boolean" minOccurs="0" default="false" maxOccurs="1">
```

# JFrog Artifactory Documentation Displayed in the header

```
<xs:annotation>
    <xs:documentation source="description">
        User authentication is required when accessing the repository. An anonymous request
        will display an HTTP 401 error. This is also enforced when aggregated repositories
        support anonymous requests.
    </xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="debianDefaultArchitectures" type="xs:string" default="i386,amd64" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            <![CDATA[
The default architectures which Debian downloads are served.
For example, when resolving distributions from remote repository, Artifactory will resolve the binary-i386
and binary-amd64 Packages files and Release files accordingly.
]]>
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="debianOptionalIndexCompressionFormats" type="FormatType" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            List of optional index compression formats for debian repository.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="useNamespaces" type="xs:boolean" minOccurs="0" default="false">
    <xs:annotation>
        <xs:documentation source="description">
            Defines whether namespaces should be used when creating a merged index.yaml file
            for a Helm virtual repository.
            The namespaces could be used for the attributes: name, url
        </xs:documentation>
    </xs:annotation>
</xs:element>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<xs:complexType name="P2ConfigurationType">
    <xs:sequence>
        <xs:element name="urls" type="P2UrlsType" maxOccurs="1" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    A set of local and remote P2 repository URLs to aggregate under this virtual repository.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="ExternalDependenciesConfigType">
    <xs:sequence>
        <xs:element name="enabled" type="xs:boolean" minOccurs="1" maxOccurs="1" default="false">
            <xs:annotation>
                <xs:documentation source="description">
                    Enable filtering of external dependencies
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="patterns" type="ExternalDependenciesPatternsType" minOccurs="0" maxOccurs="1">
            <xs:annotation>
                <xs:documentation source="description">
                    A set of ant patterns to match for external dependencies.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="remoteRepo" type="xs:IDREF" minOccurs="0" maxOccurs="1">
            <xs:annotation>
                <xs:documentation source="description">
                    Remote repo used to cache external dependencies
                </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="ExternalDependenciesPatternsType">
    <xs:sequence>
        <xs:element name="pattern" type="xs:string" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
                <xs:documentation source="description">
                    The pattern to match for external dependency
                </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="VirtualCacheConfigType">
    <xs:sequence>
        <xs:element name="virtualRetrievalCachePeriodSecs" type="xs:long" default="600" minOccurs="0">
            <xs:annotation>
```

# JFrog Artifactory Documentation Displayed in the header

```
<xs:documentation source="description">
    The number of seconds to cache artifact look-up results.
    A value of 0 indicates no caching.
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="NuGetConfigurationType">
    <xs:sequence>
        <xs:element name="feedContextPath" type="xs:string" default="api/v2" minOccurs="1">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        The context path prefix through which the NuGet feeds are served.
                        For example, the NuGet Gallery feed URL is 'https://nuget.org/api/v2', so the repository URL should
                        be configured as 'https://nuget.org' and the feed context path should be configured as 'api/v2'.
                    ]]>
            </xs:documentation>
        </xs:annotation>
        </xs:element>
        <xs:element name="downloadContextPath" type="xs:string" default="api/v2/package" minOccurs="1">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        The context path prefix through which NuGet downloads are served.
                        For example, the NuGet Gallery download URL is 'https://nuget.org/api/v2/package', so the repository
                        URL should be configured as 'https://nuget.org' and the download context path should be configured
                        as 'api/v2/package'.
                    ]]>
            </xs:documentation>
        </xs:annotation>
        </xs:element>
        <xs:element name="v3FeedUrl" type="xs:string" default="https://api.nuget.org/v3/index.json" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        The url to the nuget v3 feed.
                        For example the feed url for the official nuget.org repo is (also the default value):
                        "https://api.nuget.org/v3/index.json"
                    ]]>
            </xs:documentation>
        </xs:annotation>
        </xs:element>
        <xs:element name="symbolServerUrl" type="xs:string" default="https://symbols.nuget.org/download/symbols" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        The url to symbol server of nuget packages.
                    ]]>
            </xs:documentation>
        </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="PypiConfigurationType">
    <xs:sequence>
        <xs:element name="indexContextPath" type="xs:string" default="simple/" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        The context path to the remote server's simple index.
                        For example, for 'https://pypi.python.org/' it is 'simple/'.
                    ]]>
            </xs:documentation>
        </xs:annotation>
        </xs:element>
        <xs:element name="packagesContextPath" type="xs:string" default="packages/" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        The context path to where packages are stored at the remote server.
                        For example, for 'https://pypi.python.org/' it is 'packages/'.
                    ]]>
            </xs:documentation>
        </xs:annotation>
        </xs:element>
        <xs:element name="pyPIRegistryUrl" type="xs:string" default="https://pypi.org" minOccurs="1">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        The PyPI registry URL to communicate with.
                        Usually should be left as a default for 'https://pypi.org' unless the remote
                        is an Artifactory instance of which the value should match the remote repository URL.
                    ]]>
            </xs:documentation>
        </xs:annotation>
        </xs:element>
        <xs:element name="repositorySuffix" type="xs:string" default="simple" minOccurs="1">
            <xs:annotation>
                <xs:documentation source="description">
```

# JFrog Artifactory Documentation Displayed in the header

```
<![CDATA[  
The PyPI registry repository suffix  
Usually should be left as a default for 'simple' unless the remote  
is a PyPI server that has custom suffix, like +simple in DevPI  
]]>  
</xs:documentation>  
</xs:annotation>  
</xs:element>  
</xs:sequence>  
</xs:complexType>  
  
<xs:complexType name="BowerConfigurationType">  
<xs:sequence>  
<xs:element name="bowerRegistryUrl" type="xs:string" default="https://bower.herokuapp.com" minOccurs="1">  
<xs:annotation>  
<xs:documentation source="description">  
<![CDATA[  
The Bower registry URL to communicate with.  
Usually should be left as a default for 'https://bower.herokuapp.com' unless the remote  
is an Artifactory instance of which the value should match the remote repository URL.  
]]>  
</xs:documentation>  
</xs:annotation>  
</xs:element>  
</xs:sequence>  
</xs:complexType>  
  
<xs:complexType name="ComposerConfigurationType">  
<xs:sequence>  
<xs:element name="composerRegistryUrl" type="xs:string" default="https://packagist.org" minOccurs="1">  
<xs:annotation>  
<xs:documentation source="description">  
<![CDATA[  
The Composer registry URL to communicate with.  
Usually should be left as a default for 'https://packagist.org' unless the remote  
is an Artifactory instance of which the value should match the remote repository URL.  
]]>  
</xs:documentation>  
</xs:annotation>  
</xs:element>  
</xs:sequence>  
</xs:complexType>  
  
<xs:complexType name="TerraformConfigurationType">  
<xs:sequence>  
<xs:element name="terraformRegistryUrl" type="xs:string" default="https://registry.terraform.io"  
minOccurs="1">  
<xs:annotation>  
<xs:documentation source="description">  
<![CDATA[  
The Terraform registry URL to communicate with.  
Usually should be left as a default for 'https://registry.terraform.io' unless the remote  
is an Artifactory instance of which the value should match the remote repository URL.  
]]>  
</xs:documentation>  
</xs:annotation>  
</xs:element>  
<xs:element name="terraformProvidersUrl" type="xs:string" default="https://releases.hashicorp.com"  
minOccurs="1">  
<xs:annotation>  
<xs:documentation source="description">  
<![CDATA[  
The Terraform providers URL to download from.  
Usually should be left as a default for 'https://releases.hashicorp.com' unless the remote  
is an Artifactory instance of which the value should match the remote repository URL.  
]]>  
</xs:documentation>  
</xs:annotation>  
</xs:element>  
</xs:sequence>  
</xs:complexType>  
  
<xs:complexType name="CocoaPodsConfigurationType">  
<xs:sequence>  
<xs:element name="cocoaPodsSpecsRepoUrl" type="xs:string" default="https://github.com/CocoaPods/Specs"  
minOccurs="1">  
<xs:annotation>  
<xs:documentation source="description">  
<![CDATA[  
The CocoaPods Specs Repo URL to communicate with.  
]]>  
</xs:documentation>  
</xs:annotation>  
</xs:element>  
<xs:element name="specRepoProvider" type="VcsGitType" minOccurs="0" maxOccurs="1"/>  
</xs:sequence>  
</xs:complexType>  
  
<xs:complexType name="HelmConfigurationType">  
<xs:sequence>  
<xs:element name="chartsBaseUrl" type="xs:string" minOccurs="0">  
<xs:annotation>  
<xs:documentation source="description">  
<![CDATA[  
The Helm Charts Base URL used during packages download.  
]]>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="GitConfigurationType">
  <xs:sequence>
    <xs:element name="registryUrl" type="xs:string" default="https://github.com/rust-lang/crates.io-index"
      minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation source="description">
          <![CDATA[
            The Git registry URL to communicate with.
          ]]>
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="P2UrlsType">
  <xs:sequence>
    <xs:element name="url" type="xs:string" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation source="description">
          The URL of the target P2 repository. This URL should contain a P2 metadata descriptor
          (composite, content or artifacts).
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="RepositoryRefsType">
  <xs:sequence>
    <xs:element name="repositoryRef" type="xs:IDREF" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation source="description">
          Local, remote or virtual repository reference (key).
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="FormatType">
  <xs:sequence>
    <xs:element name="debianFormat" type="xs:string" minOccurs="0" maxOccurs="3">
      <xs:annotation>
        <xs:documentation source="description">
          Optional index compression formats for debian repository
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ProxyType">
  <xs:sequence>
    <xs:element name="key" type="xs:ID">
      <xs:annotation>
        <xs:documentation source="description">
          Proxy Unique ID.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="host" type="xs:string">
      <xs:annotation>
        <xs:documentation source="description">
          Proxy host name.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="port" type="xs:int">
      <xs:annotation>
        <xs:documentation source="description">
          Proxy port number.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="username" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation source="description">
          Proxy username.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="password" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation source="description">
          Proxy password.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="ntHost" type="xs:string" minOccurs="0">

```

# JFrog Artifactory Documentation Displayed in the header

```
<xs:annotation>
  <xs:documentation source="description">
    The computer name of this machine (that is connected to the NTLM proxy).
  </xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="domain" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      Proxy domain/realm name.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="platformDefault" type="xs:boolean" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      Make this proxy the default for all platform services, Artifactory included.
      All remote repositories and internal HTTP requests will be using this proxy.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="redirectedToHosts" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      An optional list of newline or comma separated host names to which the proxy may redirect
      requests.
      The credentials of the proxy are reused by requests redirected to any of these hosts.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="services" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      An optional list of newline or comma separated services to which the proxy be the default of.
    </xs:documentation>
  </xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="ReverseProxyType">
  <xs:sequence>
    <xs:element name="key" type="xs:ID">
      <xs:annotation>
        <xs:documentation source="description">
          Reverse Proxy Unique ID.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="webServerType" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation source="description">
          The web server type for this reverse proxy
        </xs:documentation>
      </xs:annotation>
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="direct"/>
          <xs:enumeration value="nginx"/>
          <xs:enumeration value="apache"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <!-- nullable path and server name - since the server conf update might do that,
    when docker repo path prefix is enabled. Though we try to prevent that in the validation.
    -->
    <xs:element name="artifactoryAppContext" type="xs:string" nillable="true"/>
    <xs:element name="publicAppContext" type="xs:string" nillable="true"/>
    <xs:element name="serverName" type="xs:string" nillable="true"/>
    <xs:element name="serverNameExpression" type="xs:string" nillable="true"/>
    <xs:element name="sslType" minOccurs="0" maxOccurs="1">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="self-signed"/>
          <xs:enumeration value="ca-signed"/>
          <xs:enumeration value="no-ssl"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="sslCertificate" type="xs:string" nillable="true"/>
    <xs:element name="sslKey" type="xs:string" nillable="true"/>
    <xs:element name="dockerReverseProxyMethod" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation source="description">
          method to use for reverse proxy
        </xs:documentation>
      </xs:annotation>
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="path"/>
          <xs:enumeration value="portPerRepo"/>
          <xs:enumeration value="subDomain"/>
          <xs:enumeration value="noValue"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
</xs:simpleType>
</xs:element>
<xs:element name="useHttps" type="xs:boolean" />
<xs:element name="useHttp" type="xs:boolean" />
<xs:element name="sslPort" type="xs:int"/>
<xs:element name="httpPort" type="xs:int"/>
<xs:element name="reverseProxyRepositories" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            A set of reusable reverse proxy repositories.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:complexType>
    <xs:sequence>
        <xs:element name="reverseProxyRepoConfigs" type="ReverseProxyRepoConfigType" maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="artifactoryServerName" type="xs:string" nillable="true" />
<xs:element name="upStreamName" type="xs:string" nillable="true"/>
<xs:element name="artifactoryPort" type="xs:int"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="ReverseProxyRepoConfigType">
    <xs:sequence>
        <xs:element name="repoRef" type="xs:string"/>
        <xs:element name="serverName" type="xs:string"/>
        <xs:element name="port" type="xs:int"/>
    </xs:sequence>
</xs:complexType>
</xs:element>

<xs:complexType name="BackupsType">
    <xs:sequence>
        <xs:element name="backup" type="BackupType" maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="BackupType">
    <xs:sequence>
        <xs:element name="key" type="xs:ID">
            <xs:annotation>
                <xs:documentation source="description">
                    Backup Unique ID.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="enabled" type="xs:boolean" minOccurs="0" default="true">
            <xs:annotation>
                <xs:documentation source="description">
                    Determines whether this backup is enabled or disabled.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="dir" type="xs:string" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        The directory to which backup local repository data as files.
                        The default is $ARTIFACTORY_HOME/backup/[backup_key]
                    ]]>
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="cronExp" type="xs:string">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        The Cron expression by which backup frequency is determined. For detailed information see: <a href="http://www.quartz-scheduler.org/docs/tutorials/crontrigger.html" target="_blank">The CronTrigger Tutorial</a>
                    ]]>
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="retentionPeriodHours" type="xs:int" default="168" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    The maximum number of hours to keep old backups in the destination dir.
                    Marking the "Incremental" checkbox, indicates that backups are incrementally (delta only)
                    written to the same directory: ${backupDir}/current. This "in place" backup type is suitable
                    for file-system based backup support, and cleanup of old backups is inactive in this mode.
                    The default is 168 hours = 7 days.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="createArchive" type="xs:boolean" minOccurs="0" default="false">
            <xs:annotation>
                <xs:documentation source="description">
                    Determines whether the backup output should be a zip archive or a
                    directory (the default).
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="excludedRepositories" minOccurs="0">
            <xs:annotation>
```

# JFrog Artifactory Documentation Displayed in the header

```
<xs:documentation source="description">
    A set of repository references to exclude from backup.
</xs:documentation>
</xs:annotation>
<xs:complexType>
    <xs:sequence>
        <xs:element name="repositoryRef" type="xs:string" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
                <xs:documentation source="description">
                    Local, remote or federated repository key.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="sendMailOnError" type="xs:boolean" minOccurs="0" default="true">
    <xs:annotation>
        <xs:documentation source="description">
            Send email notifications to admin users if errors are encountered during the backup process
            (requires properly configured email settings and valid email addresses for admin users).
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="excludeNewRepositories" type="xs:boolean" minOccurs="0" default="false">
    <xs:annotation>
        <xs:documentation source="description">
            Automatically exclude new repositories from the backup.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="precalculate" type="xs:boolean" minOccurs="0" default="false">
    <xs:annotation>
        <xs:documentation source="description">
            Precalculate estimated backup size before starting backup.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="exportMissionControl" type="xs:boolean" minOccurs="0" default="false">
    <xs:annotation>
        <xs:documentation source="description">
            Export mission control.
        </xs:documentation>
    </xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="IndexerType">
    <xs:annotation>
        <xs:documentation source="description">
            Configuration for the indexer creating Maven indexes downloadable by IDEs.
        </xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="enabled" type="xs:boolean" default="false" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    Determines whether the indexer service is enabled and should be run.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="cronExp" type="xs:string" default="0 23 5 * * ?" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        The Cron expression by which indexer frequency is determined to recalculate
                        new Maven indexes on selected repositories. For detailed information see: <a href="http://www.quartz-scheduler.org/docs/tutorials/crontrigger.html"
target="_blank">The CronTrigger Tutorial</a>
                    ]]>
                </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:element name="includedRepositories" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            A set of repository references to index.
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="repositoryRef" type="xs:string" minOccurs="0" maxOccurs="unbounded">
                <xs:annotation>
                    <xs:documentation source="description">
                        Repository key.
                    </xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
<xs:complexType name="AddonsType">
  <xs:sequence>
    <xs:element name="showAddonsInfo" type="xs:boolean" minOccurs="0" maxOccurs="1" default="true">
      <xs:annotation>
        <xs:documentation source="description">
          When selected, info about available Add-ons is displayed in the Artifactory user interface.
          Enabling add-ons info display resets any user-specific info-hiding preference.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="showAddonsInfoCookie" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation source="description">
          A variable used for calculating the path for a user cookie storing their personal add-on
          info showing preferences.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="XrayType">
  <xs:sequence>
    <xs:element name="enabled" type="xs:boolean" default="true" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation source="description">
          The activity state of the configuration.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="baseUrl" type="xs:string" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation source="description">
          The base url of xray server.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="user" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation source="description">
          The xray user.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="password" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation source="description">
          The xray password.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="artifactoryId" type="xs:string" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation source="description">
          The artifactoryId.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="xrayId" type="xs:string" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation source="description">
          The Xray ID.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="allowOperationsXrayUnavailable" type="xs:boolean" default="false" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation source="description">
          The configuration that states what to do when Xray is offline.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="allowBlockedArtifactsDownload" type="xs:boolean" default="false" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation source="description">
          The configuration that states we need to avoid blocking.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="blockUnscannedTimeoutSeconds" type="xs:int" default="60" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation source="description">
          When a repository is configured to block downloads of unscanned artifacts,
          this setting will make every download request connection to remain open for the time configured (in seconds),
          allowing Xray sufficient time to scan the artifact and then return the artifact or block it based on scan results.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="blockUnfinishedScansTimeoutSeconds" type="xs:int" default="600" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation source="description">
          The required waiting time of scanning a Release Bundle in XRay before being considered as stuck in scanning status.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

# JFrog Artifactory Documentation Displayed in the header

```
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="RepoXrayConfigType">
<xs:sequence>
<xs:element name="enabled" type="xs:boolean" default="false" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
When enabled, this repository will send index events to Xray.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="dataTtl" type="xs:int" minOccurs="0" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
The field will indicate for how long Xray data for artifacts in this repository shall be saved.
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="RepoDownloadRedirectConfigType">
<xs:sequence>
<xs:element name="enabled" type="xs:boolean" default="false" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
When enabled, this repository will respond with HTTP 302 for every download request.
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="RepoCdnRedirectConfigType">
<xs:sequence>
<xs:element name="enabled" type="xs:boolean" default="false" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
When enabled, this repository will respond with HTTP 302 to CDN for every download request.
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="CargoConfigType">
<xs:sequence>
<xs:element name="cargoAnonymousAccess" type="xs:boolean" default="false" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
When enabled, will allow anonymous access to crate downloads and search, as the cargo client does not send credentials in those cases
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="cargoInternalIndex" type="xs:boolean" default="false" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
When enabled, cargo will use HTTP indexes instead of default git one
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="MailServerType">
<xs:sequence>
<xs:element name="enabled" type="xs:boolean" default="true" minOccurs="0" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
The activity state of the configuration.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="host" type="xs:string" minOccurs="0" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
The host name of the mail server.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="port" type="xs:int" default="25" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
The mail server port.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="username" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
The authentication username.
</xs:annotation>
```

# JFrog Artifactory Documentation Displayed in the header

```
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="password" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      The authentication password.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="from" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      The "from" address header to use in all outgoing mails. Can be left blank.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="subjectPrefix" type="xs:string" minOccurs="0" default="[JFrog]">
  <xs:annotation>
    <xs:documentation source="description">
      A prefix to use for the subject of all outgoing mails.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="tls" type="xs:boolean" minOccurs="0" default="false">
  <xs:annotation>
    <xs:documentation source="description">
      Determines whether to use transport layer security when connecting to the mail server.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="ssl" type="xs:boolean" minOccurs="0" default="false">
  <xs:annotation>
    <xs:documentation source="description">
      Determines whether to use SSL when connecting to the mail server.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="artifactoryUrl" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      The Artifactory URL to use in all outgoing mails to denote links to Artifactory. Can be left blank.
    </xs:documentation>
  </xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="DownloadRedirectConfigType">
  <xs:annotation>
    <xs:documentation source="description">
      The default download redirect configuration that will be used by Artifactory in cases where a repository is configured to allow download redirect to cloud storage.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="fileMinimumSize" type="xs:long" default="1" minOccurs="0">
      <xs:annotation>
        <xs:documentation source="description">
          The minimum file-size in MB of artifacts to be redirected with signed url instead of direct download.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="RepositoriesGeneralType">
  <xs:sequence>
    <xs:element name="helm" type="helmType" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation source="description">
          <![CDATA[
            General repositories settings, specific to helm.
          ]]>
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="helmType">
  <xs:sequence>
    <xs:element name="forceOciFirstUiFlow" type="xs:boolean" minOccurs="0" default="true">
      <xs:annotation>
        <xs:documentation source="description">
          <![CDATA[
            Define what is the default repository settings, whether to see first UI OCI or Legacy.
          ]]>
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

# JFrog Artifactory Documentation Displayed in the header

```
<xs:complexType name="SecurityType">
  <xs:sequence>
    <xs:element name="hideUnauthorizedResources" type="xs:boolean" minOccurs="0" default="false">
      <xs:annotation>
        <xs:documentation source="description">
          When marked, Artifactory hides the existence of unauthorized resources by sending a 404
          response (not found) to requests for resources that are not accessible by the user. Otherwise,
          the response implies that the resource exists but is protected - by requesting authentication
          for anonymous requests (401) or by denying an authenticated request for unauthorized users.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="passwordSettings" type="PasswordSettingsType" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation source="description">
          <![CDATA[
            Password policy settings (credentials encryption etc.).
          ]]&gt;
        &lt;/xs:documentation&gt;
      &lt;/xs:annotation&gt;
    &lt;/xs:element&gt;
    &lt;xs:element name="oauthSettings" type="oauthSettingsType" minOccurs="0" maxOccurs="1"&gt;
      &lt;xs:annotation&gt;
        &lt;xs:documentation source="description"&gt;
          <![CDATA[
            Security settings specific to OAuth.
          ]]&gt;
        &lt;/xs:documentation&gt;
      &lt;/xs:annotation&gt;
    &lt;/xs:element&gt;
    &lt;xs:element name="centralOAuthProviders" type="centralOAuthProvidersType" minOccurs="0" maxOccurs="1"&gt;
      &lt;xs:annotation&gt;
        &lt;xs:documentation source="description"&gt;
          <![CDATA[
            Security settings specific to OAuth.
          ]]&gt;
        &lt;/xs:documentation&gt;
      &lt;/xs:annotation&gt;
    &lt;/xs:element&gt;
    &lt;xs:element name="signingKeysSettings" type="SigningKeysSettingsType" minOccurs="0" maxOccurs="1"&gt;
      &lt;xs:annotation&gt;
        &lt;xs:documentation source="description"&gt;
          <![CDATA[
            Security settings specific to GPG signing
          ]]&gt;
        &lt;/xs:documentation&gt;
      &lt;/xs:annotation&gt;
    &lt;/xs:element&gt;
    &lt;xs:element name="sshServerSettings" type="SshServerSettingsType" minOccurs="0" maxOccurs="1"&gt;
      &lt;xs:annotation&gt;
        &lt;xs:documentation source="description"&gt;
          <![CDATA[
            Security settings specific to SSH.
          ]]&gt;
        &lt;/xs:documentation&gt;
      &lt;/xs:annotation&gt;
    &lt;/xs:element&gt;
    &lt;xs:element name="buildGlobalBasicReadAllowed" type="xs:boolean" minOccurs="0" default="false"&gt;
      &lt;xs:annotation&gt;
        &lt;xs:documentation source="description"&gt;
          Determines whether users can access basic build related information.
        &lt;/xs:documentation&gt;
      &lt;/xs:annotation&gt;
    &lt;/xs:element&gt;
    &lt;xs:element name="buildGlobalBasicReadForAnonymous" type="xs:boolean" minOccurs="0" default="false"&gt;
      &lt;xs:annotation&gt;
        &lt;xs:documentation source="description"&gt;
          Determines whether anonymous user can access basic build related information.
        &lt;/xs:documentation&gt;
      &lt;/xs:annotation&gt;
    &lt;/xs:element&gt;
  &lt;/xs:sequence&gt;
&lt;/xs:complexType&gt;

&lt;xs:complexType name="PasswordSettingsType"&gt;
  &lt;xs:sequence&gt;
    &lt;xs:element name="expirationPolicy" type="PasswordExpirationPolicyType" minOccurs="0" maxOccurs="1"&gt;
      &lt;xs:annotation&gt;
        &lt;xs:documentation source="description"&gt;
          <![CDATA[
            Password expiration policy.
          ]]&gt;
        &lt;/xs:documentation&gt;
      &lt;/xs:annotation&gt;
    &lt;/xs:element&gt;
    &lt;xs:element name="resetPolicy" type="PasswordResetPolicyType" minOccurs="0" maxOccurs="1"&gt;
      &lt;xs:annotation&gt;
        &lt;xs:documentation source="description"&gt;
          <![CDATA[
            Password reset protection policy.
          ]]&gt;
        &lt;/xs:documentation&gt;
      &lt;/xs:annotation&gt;
    &lt;/xs:element&gt;
  &lt;/xs:sequence&gt;
&lt;/xs:complexType&gt;</pre>
```

# JFrog Artifactory Documentation Displayed in the header

```
</xs:sequence>
</xs:complexType>

<xs:complexType name="PasswordExpirationPolicyType">
    <xs:sequence>
        <xs:element name="enabled" type="xs:boolean" minOccurs="0" maxOccurs="1" default="false">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        Whether user password can expire.
                    ]]>
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="passwordMaxAge" type="xs:int" minOccurs="0" maxOccurs="1">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        The number of days for password to get expired.
                    ]]>
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="notifyByEmail" type="xs:boolean" minOccurs="0" maxOccurs="1" default="true">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        Whether users should be notified by email about password expiration.
                    ]]>
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="currentPasswordValidFor" type="xs:int" minOccurs="0" maxOccurs="1">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        Deprecated. Current authenticated entity password valid for
                    ]]>
                </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="PasswordResetPolicyType">
    <xs:sequence>
        <xs:element name="enabled" type="xs:boolean" minOccurs="0" maxOccurs="1" default="true">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        Whether to protect against password reset attacks.
                    ]]>
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="maxAttemptsPerAddress" type="xs:int" minOccurs="0" maxOccurs="1">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        The maximum number of password reset attempts per address.
                    ]]>
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="timeToBlockInMinutes" type="xs:int" minOccurs="0" maxOccurs="1">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        The time in minutes to block password reset attempts per address.
                    ]]>
                </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="SshServerSettingsType">
    <xs:sequence>
        <xs:element name="enableSshServer" type="xs:boolean" minOccurs="0" default="false">
            <xs:annotation>
                <xs:documentation source="description">
                    <![CDATA[
                        SSH n stuff
                    ]]>
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="sshServerPort" type="xs:int" default="1337" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    SSH server port number.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
</xs:sequence>
</xs:complexType>
<xs:complexType name="oauthSettingsType">
  <xs:sequence>
    <xs:element name="enableIntegration" type="xs:boolean" minOccurs="0" maxOccurs="1" default="false">
      <xs:annotation>
        <xs:documentation source="description">
          <![CDATA[
            Enable security integration with OAuth.
          ]]>
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="allowUserToAccessProfile" type="xs:boolean" minOccurs="0" default="false">
      <xs:annotation>
        <xs:documentation source="description">
          <![CDATA[
            Auto created users will have access to their profile page and will be able to perform actions
            such as generate API key.
          ]]>
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="persistUsers" type="xs:boolean" minOccurs="0" maxOccurs="1" default="false">
      <xs:annotation>
        <xs:documentation source="description">
          <![CDATA[
            Persist new users.
          ]]>
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="defaultNpm" type="xs:string" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation source="description">
          <![CDATA[
            Default NPM provider.
          ]]>
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="oauthProvidersSettings" type="oauthProvidersSettingsType" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation source="description">
          <![CDATA[
            Security settings specific to LDAP Groups.
          ]]>
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="centralOAuthProvidersType">
  <xs:sequence>
    <xs:element name="enabled" type="xs:boolean" minOccurs="0" maxOccurs="1" default="false">
      <xs:annotation>
        <xs:documentation source="description">
          <![CDATA[
            Enable security integration with central OAuth providers.
          ]]>
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:sequence>
      <xs:element name="enabledProviderNames" type="xs:string" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation source="description">
            <![CDATA[
              Enabled central OAuth providers.
            ]]>
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="oauthProvidersSettingsType">
  <xs:sequence>
    <xs:element name="oauthProvidersSettings" type="oauthProviderSettingsType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="oauthProviderSettingsType">
  <xs:sequence>
    <xs:element name="name" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation source="description">
          <![CDATA[
            Provider unique name.
          ]]>
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="enabled" type="xs:boolean" minOccurs="0" default="false">
      <xs:annotation>
```

# JFrog Artifactory Documentation Displayed in the header

```
<xs:documentation source="description">
  <![CDATA[
    Provider type.
  ]]>
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="providerType" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      <![CDATA[
        Provider type.
      ]]>
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="id" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      <![CDATA[
        Provider id.
      ]]>
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="secret" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      <![CDATA[
        Provider secret.
      ]]>
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="apiUrl" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      <![CDATA[
        Provider url.
      ]]>
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="authUrl" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      <![CDATA[
        Provider authenticationURL.
      ]]>
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="tokenUrl" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      <![CDATA[
        Provider token URL.
      ]]>
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="basicUrl" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      <![CDATA[
        Provider basic URL.
      ]]>
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="domain" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      <![CDATA[
        Provider domain.
      ]]>
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="central" type="xs:boolean" minOccurs="0" default="false">
  <xs:annotation>
    <xs:documentation source="description">
      <![CDATA[
        Whether the provider scope is central or private.
      ]]>
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="pkce" type="xs:boolean" minOccurs="0" default="false">
  <xs:annotation>
    <xs:documentation source="description">
      <![CDATA[
        Whether PKCE is enabled.
      ]]>
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="SigningKeysSettingType">
<xs:sequence>
<xs:element name="keyStorePassword" type="xs:string" minOccurs="0" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
<![CDATA[ Password for the Key Store. ]]>
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="PropertySetsType">
<xs:sequence>
<xs:element name="propertySet" type="PropertySetType" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="SystemPropertiesType">
<xs:sequence>
<xs:element name="systemProperty" type="SystemPropertyType" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="PropertySetType">
<xs:sequence>
<xs:element name="name" type="xs:ID">
<xs:annotation>
<xs:documentation source="description">
A unique name identifying the property set.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="visible" type="xs:boolean">
<xs:annotation>
<xs:documentation source="description">
Whether this property set is visible in the properties management user interface, or is
intended for internal use.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="properties" type="PropertiesType" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
The list of properties grouped under this property set.
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="SystemPropertyType">
<xs:sequence>
<xs:element name="key" type="xs:ID">
<xs:annotation>
<xs:documentation source="description">
A unique name of the system property.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="value" type="xs:ID">
<xs:annotation>
<xs:documentation source="description">
A list of predefined values to select from for the property value.
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="PropertiesType">
<xs:sequence>
<xs:element name="property" type="PropertyType" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name=".PropertyType">
<xs:sequence>
<xs:element name="name" type="xs:string">
<xs:annotation>
<xs:documentation source="description">
A unique name identifying the property.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="closedPredefinedValues" type="xs:boolean">
<xs:annotation>
<xs:documentation source="description">
Whether the values of the property shall come from a predefined closed list of values.
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="multipleChoice" type="xs:boolean">
<xs:annotation>
    <xs:documentation source="description">
        Whether the property supports multiple values.
    </xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="predefinedValues" type="PredefinedValuesType" minOccurs="0">
<xs:annotation>
    <xs:documentation source="description">
        A list of predefined values to select from for the property value(s).
    </xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="PredefinedValuesType">
<xs:sequence>
    <xs:element name="predefinedValue" type="PredefinedValueType" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="PredefinedValueType">
<xs:sequence>
    <xs:element name="value" type="xs:string">
        <xs:annotation>
            <xs:documentation source="description">
                The actual value.
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="defaultValue" type="xs:boolean">
        <xs:annotation>
            <xs:documentation source="description">
                Whether or not this value is a default/pre-selected one (multiple choice properties support
                multiple default values).
            </xs:documentation>
        </xs:annotation>
    </xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="RepoLayoutType">
<xs:sequence>
    <xs:element name="name" type="xs:ID">
        <xs:annotation>
            <xs:documentation source="description">
                A unique name identifying the layout.
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="artifactPathPattern" type="xs:string">
        <xs:annotation>
            <xs:documentation source="description">
                <![CDATA[(&lt;HTML&gt;
                &lt;div class="legend-surrounding-text"&gt;
                    The path pattern that matches the storage path of artifacts.&lt;br/&gt;
                    The path pattern is composed of literals (including path separators) and any of the following tokens:&lt;br/&gt;
                    &lt;i&gt;org, orgPath, baseRev, fileIntegRev, folderIntegRev, module, classifier, ext, type.&lt;/i&gt;
                &lt;/div&gt;
                &lt;table class="legend"&gt;
                    &lt;thead&gt;
                        &lt;tr&gt;
                            &lt;th&gt;Token&lt;/th&gt;
                            &lt;th&gt;Description&lt;/th&gt;
                            &lt;th&gt;Example Value&lt;/th&gt;
                        &lt;/tr&gt;
                    &lt;/thead&gt;
                    &lt;tbody&gt;
                        &lt;tr&gt;
                            &lt;td class="token"&gt;[org]&lt;/td&gt;
                            &lt;td class="desc"&gt;
                                Identifies the artifact's organization.
                                Equivalent to Ivy's 'organization'.
                            &lt;/td&gt;
                            &lt;td class="example"&gt;
                                &lt;span class="e-value"&gt;'org.apache.derby'&lt;/span&gt;
                            &lt;/td&gt;
                        &lt;/tr&gt;
                        &lt;tr&gt;
                            &lt;td class="token"&gt;[orgPath]&lt;/td&gt;
                            &lt;td class="desc"&gt;
                                Identifies the artifact's organization where dots ('.') are replaced by path separators ('/'). Equivalent to Maven's 'groupId'.
                            &lt;/td&gt;
                            &lt;td class="example"&gt;
                                &lt;span class="e-value"&gt;'org/apache/derby'&lt;/span&gt;
                            &lt;/td&gt;
                        &lt;/tr&gt;
                        &lt;tr&gt;
                            &lt;td class="token"&gt;[module]&lt;/td&gt;
                        &lt;/tr&gt;
                    &lt;/tbody&gt;
                &lt;/table&gt;
            &lt;/xs:documentation&gt;
        &lt;/xs:annotation&gt;
    &lt;/xs:element&gt;
&lt;/xs:sequence&gt;
&lt;/xs:complexType&gt;</pre>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
<td class="desc">
    Identifies the artifact's module.
</td>
<td class="example">
    <span class="e-value">'hibernate'</span>
</td>
</tr>
<tr>
    <td class="token">[baseRev]</td>
    <td class="desc">
        Identifies the base revision part of the artifact version, excluding any integration information
    </td>
    <td class="example">
        <span class="e-value">'2.1.1'</span> which may be part of the full version '2.1.1-SNAPSHOT' or '2.1.1-build.1099'.
    </td>
</tr>
<tr>
    <td class="token">[folderItegRev]</td>
    <td class="desc">
        Identifies the integration revision part used in folder names in the artifact's path, excluding the base revision.
    </td>
    <td class="example">
        <span class="e-value">'SNAPSHOT'</span> or <span class="e-value">'build.1099'</span> which may be part of the folder name '2.1.1-SNAPSHOT' or '2.1.1-build.1099', respectively.
    </td>
</tr>
<tr>
    <td class="token">[fileItegRev]</td>
    <td class="desc">
        Identifies the integration revision part in the artifact's file name, excluding the base revision.
    </td>
    <td class="example">
        <span class="e-value">'20110113.145509-8'</span> or <span class="e-value">'build.1099'</span> which may be part of the file name '2.1.1-20110113.145509-8' or '2.1.1-build.1099', respectively.
    </td>
</tr>
<tr>
    <td class="token">[classifier]</td>
    <td class="desc">
        Identifies the artifact's classifier.
    </td>
    <td class="example">
        <span class="e-value">'sources'</span>
    </td>
</tr>
<tr>
    <td class="token">[ext]</td>
    <td class="desc">
        Identifies the artifact's extension.
    </td>
    <td class="example">
        <span class="e-value">'jar'</span>
    </td>
</tr>
<tr>
    <td class="token">[type]</td>
    <td class="desc">
        Identifies the artifact's type. Typically used when the artifact's extension cannot be reused as the artifact's type.
    </td>
    <td class="example">
        <span class="e-value">'javadoc'</span> (that can have the 'jar' extension) or 'ivy' (that can have the 'xml' extension).
    </td>
</tr>
<tr>
    <td class="token">[customToken&lt;regex&gt;]</td>
    <td class="desc">
        A custom regex token. Can be used to create a new type of token when the provided defaults don't suffice.
    </td>
    <td class="example">
        <span class="e-value">'[timestamp<\d{8}]'</span> a new timestamp custom token that has 8 digits.
    </td>
</tr>
</tbody>
</table>
<div class="legend-surrounding-text">
    For Artifactory to be able to construct meaningful module information the following tokens have to be used at the minimum:<br/>
    <i>org or orgPath, baseRev and module.</i>
</div>
]]>
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="distinctiveDescriptorPathPattern" type="xs:boolean">
<xs:annotation>
    <xs:documentation source="description">
        <![CDATA[(HTML)
<p>
            Check this if the repository layout uses separate path patterns for artifacts and descriptors
            (such as Ivy descriptor files). Otherwise, the artifact path pattern will be used to match
            descriptors.
        </p>
<p>
            The descriptor path pattern matches the storage path of module descriptors and uses the same
            format as of the artifact path pattern.
        </p>
    </xs:documentation>

```

# JFrog Artifactory Documentation

## Displayed in the header

```
</p>
<table class="legend">
  <thead>
    <tr>
      <th>Token</th>
      <th>Description</th>
      <th>Example Value</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td class="token">[org]</td>
      <td class="desc">
        Identifies the artifact's organization.
        Equivalent to Ivy's 'organization'.
      </td>
      <td class="example">
        <span class="e-value">'org.apache.derby'</span>
      </td>
    </tr>
    <tr>
      <td class="token">[orgPath]</td>
      <td class="desc">
        Identifies the artifact's organization where dots ('.') are replaced by path separators ('/'). Equivalent to Maven's 'groupId'.
      </td>
      <td class="example">
        <span class="e-value">'org/apache/derby'</span>
      </td>
    </tr>
    <tr>
      <td class="token">[baseRev]</td>
      <td class="desc">
        Identifies the base revision part of the artifact version, excluding any integration information
      </td>
      <td class="example">
        <span class="e-value">'2.1.1'</span> which may be part of the full version '2.1.1-SNAPSHOT' or '2.1.1-build.1099'.
      </td>
    </tr>
    <tr>
      <td class="token">[fileItegRev]</td>
      <td class="desc">
        Identifies the integration revision part in the artifact's file name, excluding the base revision.
      </td>
      <td class="example">
        <span class="e-value">'20110113.145509-8'</span> or <span class="e-value">'build.1099'</span> which may be part of the file name '2.1.1-20110113.145509-8' or '2.1.1-build.1099', respectively.
      </td>
    </tr>
    <tr>
      <td class="token">[folderItegRev]</td>
      <td class="desc">
        Identifies the integration revision part used in folder names in the artifact's path, excluding the base revision.
      </td>
      <td class="example">
        <span class="e-value">'SNAPSHOT'</span> or <span class="e-value">'build.1099'</span> which may be part of the folder name '2.1.1-SNAPSHOT' or '2.1.1-build.1099', respectively.
      </td>
    </tr>
    <tr>
      <td class="token">[module]</td>
      <td class="desc">
        Identifies the artifact's module.
      </td>
      <td class="example">
        <span class="e-value">'hibernate'</span>
      </td>
    </tr>
    <tr>
      <td class="token">[classifier]</td>
      <td class="desc">
        Identifies the artifact's classifier.
      </td>
      <td class="example">
        <span class="e-value">'sources'</span>
      </td>
    </tr>
    <tr>
      <td class="token">[ext]</td>
      <td class="desc">
        Identifies the artifact's extension.
      </td>
      <td class="example">
        <span class="e-value">'jar'</span>
      </td>
    </tr>
    <tr>
      <td class="token">[type]</td>
      <td class="desc">
        Identifies the artifact's type. Typically used when the artifact's extension cannot be reused as the artifact's type.
      </td>
      <td class="example">
        <span class="e-value">'javadoc'</span> (that can have the 'jar' extension) or 'ivy' (that can have the 'xml' extension).
      </td>
    </tr>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
<tr>
    <td class="token">[customToken&lt;regex&gt;]</td>
    <td class="desc">
        A custom regex token. Can be used to create a new type of token when the provided defaults don't suffice.
    </td>
    <td class="example">
        <span class="e-value">[timestamp<\d{8}>]</span> a new timestamp custom token that has 8 digits.
    </td>
</tr>
</tbody>
]
]

</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="descriptorPathPattern" type="xs:string" minOccurs="0">
<xs:annotation>
    <xs:documentation source="description">
        The path pattern matching the storage path of module descriptors.
        Uses the same format as the artifact path pattern.
    </xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="folderIntegrationRevisionRegExp" type="xs:string" minOccurs="0">
<xs:annotation>
    <xs:documentation source="description">
        A regular expression matching the integration revision string appearing in a folder name
        as part of the artifact's path. For example, 'SNAPSHOT', in Maven.
        Note! that you must ensure not to introduce any regexp capturing groups within this expression.
        If not applicable use '.'.
    </xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="fileIntegrationRevisionRegExp" type="xs:string" minOccurs="0">
<xs:annotation>
    <xs:documentation source="description">
        A regular expression matching the integration revision string appearing in a file name
        as part of the artifact's path. For example, 'SNAPSHOT|(?:[:0-9]{8}.[0-9]{6})-(?:[0-9]+)', in Maven.
        Note! that you must ensure not to introduce any regexp capturing groups within this expression.
        If not applicable use '.'.
    </xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="ReplicationBaseType">
<xs:sequence>
    <xs:element name="enabled" type="xs:boolean">
        <xs:annotation>
            <xs:documentation source="description">
                Enable active replication of this repository.
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="cronExp" type="xs:string" minOccurs="0">
        <xs:annotation>
            <xs:documentation source="description">
                <![CDATA[
                    The Cron expression by which replication frequency is determined. For detailed information see: <a href="http://www.quartz-scheduler.org/docs/tutorials/crontrigger.html" target="_blank">The CronTrigger Tutorial</a>
                ]]>
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="syncDeletes" type="xs:boolean">
        <xs:annotation>
            <xs:documentation source="description">
                Delete artifacts and folders that no longer exist in the source repository.
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="syncProperties" type="xs:boolean">
        <xs:annotation>
            <xs:documentation source="description">
                Include metadata properties of files and folders as part of the replication.
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="repoKey" type="xs:string" minOccurs="1">
        <xs:annotation>
            <xs:documentation source="description">
                The replicated repository key.
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="enableEventReplication" type="xs:boolean" minOccurs="0">
        <xs:annotation>
            <xs:documentation source="description">
                Replicate additions and modifications as they occur.
            </xs:documentation>
        </xs:annotation>
    </xs:element>

```

# JFrog Artifactory Documentation

## Displayed in the header

```
<xs:element name="replicationKey" type="xs:string" minOccurs="1">
    <xs:annotation>
        <xs:documentation source="description">
            Unique identification of the replication.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="checkBinaryExistenceInFilestore" type="xs:boolean" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            Indicates an attempt should be made to locate the binary directly from the filestore, if it is
            not found as a record in the database.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="includePathPrefixPattern" type="xs:string" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            An inclusion subpath pattern within the source repository to limit replication to.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="excludePathPrefixPattern" type="xs:string" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            An exclusion subpath pattern within the source repository to limit replication to.
        </xs:documentation>
    </xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="RemoteReplicationType">
    <xs:complexContent>
        <xs:extension base="ReplicationBaseType">
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

<xs:complexType name="LocalReplicationType">
    <xs:complexContent>
        <xs:extension base="ReplicationBaseType">
            <xs:sequence>
                <xs:element name="url" type="xs:string" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            The URL of the target local repository on a remote Artifactory server.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="proxyRef" type="xs:IDREF" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            Network proxy reference.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="disableProxy" type="xs:boolean" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            Turns on/off proxy usage for this repository.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="socketTimeoutMillis" type="xs:int" default="15000" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            Network timeout in milliseconds to use both for connection
                            establishment and for unanswered requests.
                            Timing out on a network operation is considered as a retrieval
                            failure.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="username" type="xs:string" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            HTTP authentication username.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="password" type="xs:string" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            HTTP authentication password.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="syncStatistics" type="xs:boolean" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation source="description">
                            Include statistics of files as part of the replication (intended mainly for Disaster Recovery).
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
</xs:element>
</xs:sequence>
</xs:extension>
</xs:complexType>
<xs:complexType name="GcConfigType">
<xs:annotation>
  <xs:documentation source="description">
    GC config.
  </xs:documentation>
</xs:annotation>
<xs:sequence>
  <xs:element name="cronExp" type="xs:string" default="* * * * ?" minOccurs="1">
    <xs:annotation>
      <xs:documentation source="description">
        <![CDATA[
          The Cron expression by which garbage collection frequency is determined. For detailed information see: <a href="http://www.quartz-scheduler.org/docs/tutorials/crontrigger.html" target="_blank">The CronTrigger Tutorial </a>
        ]]>
      </xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="CleanupConfigType">
<xs:annotation>
  <xs:documentation source="description">
    Cleanup config.
  </xs:documentation>
</xs:annotation>
<xs:sequence>
  <xs:element name="cronExp" type="xs:string" default="0 12 5 * * ?" minOccurs="1">
    <xs:annotation>
      <xs:documentation source="description">
        <![CDATA[
          The Cron expression by which artifacts cleanup frequency is determined. For detailed information see: <a href="http://www.quartz-scheduler.org/docs/tutorials/crontrigger.html" target="_blank">The CronTrigger Tutorial</a>
        ]]>
      </xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="QuotaConfigType">
<xs:annotation>
  <xs:documentation source="description">
    Quota management configuration.
  </xs:documentation>
</xs:annotation>
<xs:sequence>
  <xs:element name="enabled" type="xs:boolean" default="false" minOccurs="0">
    <xs:annotation>
      <xs:documentation source="description">
        Enable control over the size of storage space used for binaries to avoid running out of disk space.
      </xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="diskSpaceLimitPercentage" type="xs:int" default="0" minOccurs="0">
    <xs:annotation>
      <xs:documentation source="description">
        The maximum disk usage allowed by percentage, of the partition containing the binaries folder. Once this limit has been reached, deployments are rejected with a 413 error (request entity too large) and an error message is displayed in the UI (visible to admin users only). With a filesystem storage the partition checked is the one containing the '$ARTIFACTORY_HOME/data/filestore' directory. With a database BLOB storage the partition checked is the one containing the '$ARTIFACTORY_HOME/data/cache' directory.
      </xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="diskSpaceWarningPercentage" type="xs:int" default="0" minOccurs="0">
    <xs:annotation>
      <xs:documentation source="description">
        The maximum warning-level of disk usage, by percentage, of the partition containing the binaries folder. Once this limit is reached a warning is logged and a warning message is displayed in the UI (visible to admin users only).
      </xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="FolderDownloadConfigType">
<xs:annotation>
  <xs:documentation source="description">
    Configuration for the ability to download folders from the UI.
  </xs:documentation>
</xs:annotation>
<xs:sequence>
  <xs:element name="enabled" type="xs:boolean" default="false" minOccurs="1" maxOccurs="1">
    <xs:annotation>
      <xs:documentation source="description">
```

# JFrog Artifactory Documentation Displayed in the header

```
Determines whether downloading folders is allowed.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="enabledForAnonymous" type="xs:boolean" default="false" minOccurs="1" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
Determines whether downloading folders is allowed for anonymous users.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="maxDownloadSizeMb" type="xs:int" minOccurs="1" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
Max allowed size of all uncompressed files under a tree to download in a single request, in MB.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="maxFiles" type="xs:long" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
Max allowed number of files to download in a single request.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="maxConcurrentRequests" type="xs:int" minOccurs="1" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
Maximum number of folder download requests that will be treated concurrently before being queued.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="enabledEmptyDirectories" type="xs:boolean" default="false" minOccurs="0" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
Determines whether empty directories are included in folder download.
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="TrashcanConfigType">
<xs:annotation>
<xs:documentation source="description">
Configuration for the global trashcan repository.
</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="enabled" type="xs:boolean" default="true" minOccurs="1" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
Determines whether trashcan is enabled.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="allowPermDeletes" type="xs:boolean" default="false" minOccurs="0" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
Allow users with delete permission to optionally skip the trash upon deletion.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="retentionPeriodDays" type="xs:int" default="30" minOccurs="1" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
The maximum number of days to keep deleted artifacts in the trashcan.
The default is 30 days.
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="FederatedMembersType">
<xs:sequence>
<xs:element name="federatedMember" type="FederatedMemberType" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="FederatedMemberType">
<xs:sequence>
<xs:element name="url" type="xs:string" minOccurs="1" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
Member url
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="mode" type="xs:string" minOccurs="0" maxOccurs="1">
<xs:annotation>
<xs:documentation source="description">
Member replication mode bidirectional or unidirectional
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="enabled" type="xs:boolean" default="true" minOccurs="0" maxOccurs="1">
<xs:annotation>
    <xs:documentation source="description">
        When false - the local mirror pausing federating events to remote mirrors.
    </xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="VcsType">
<xs:sequence>
    <xs:element name="type">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="git"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="git" type="VcsGitType"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="VcsGitType">
<xs:sequence>
    <xs:element name="provider" minOccurs="1" maxOccurs="1">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="github"/>
                <xs:enumeration value="bitbucket"/>
                <xs:enumeration value="stash"/>
                <xs:enumeration value="oldstash"/>
                <xs:enumeration value="artifactory"/>
                <xs:enumeration value="custom"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="downloadUrl" type="xs:string" minOccurs="0">
        <xs:annotation>
            <xs:documentation source="description">
                <![CDATA[<HTML>
                    <div class="legend-surrounding-text">
                        The download pattern which matches the remote repository tarballs URLs.<br/>
                        The download pattern is composed of literals (including path separators) and placeholders for the following tokens:<br/>
                        <i>userOrg, repo, refName, fileExtension.</i><br/>
                    </div>
                    <table class="legend">
                        <thead>
                            <tr>
                                <th>Token</th>
                                <th>Description</th>
                            </tr>
                        </thead>
                        <tbody>
                            <tr>
                                <td class="token">{0}</td>
                                <td class="desc">
                                    Identifies the username or organization name.
                                </td>
                            </tr>
                            <tr>
                                <td class="token">{1}</td>
                                <td class="desc">
                                    Identifies the repository name.
                                </td>
                            </tr>
                            <tr>
                                <td class="token">{2}</td>
                                <td class="desc">
                                    Identifies the branch or tag name.
                                </td>
                            </tr>
                            <tr>
                                <td class="token">{3}</td>
                                <td class="desc">
                                    Identifies the file extension to download.
                                </td>
                            </tr>
                        </tbody>
                    </table>
                    <div class="legend-surrounding-text">
                        For example, GitHub uses the following pattern to download tarballs: https://github.com/{userOrg}/{repo}/archive/{refName}.{fileExt}<br/>
                        So to be able to download from any repository in GitHub, the composed download pattern should be {0}/{1}/archive/{2}.{3}.<br/>
                    </div>
                ]]>
            </xs:documentation>
        </xs:annotation>
    </xs:element>
</xs:sequence>
</xs:complexType>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
<xss:complexType name="replicationsConfigType">
<xss:annotation>
<xss:documentation source="description">
    Global replication configuration.
</xss:documentation>
</xss:annotation>
<xss:sequence>
    <xss:element name="blockPushReplications" type="xs:boolean" default="false" minOccurs="0">
        <xss:annotation>
            <xss:documentation source="description">
                Globally disable the push replication.
            </xss:documentation>
        </xss:annotation>
    </xss:element>
    <xss:element name="blockPullReplications" type="xs:boolean" default="false" minOccurs="0">
        <xss:annotation>
            <xss:documentation source="description">
                Globally disable the pull replication.
            </xss:documentation>
        </xss:annotation>
    </xss:element>
</xss:sequence>
</xss:complexType>

<xss:complexType name="SumoLogicConfigType">
<xss:annotation>
<xss:documentation source="description">
    Configuration for the Sumo Logic integration.
</xss:documentation>
</xss:annotation>
<xss:sequence>
    <xss:element name="enabled" type="xs:boolean" default="true" minOccurs="1" maxOccurs="1">
        <xss:annotation>
            <xss:documentation source="description">
                Determines whether Sumo Logic integration is enabled.
            </xss:documentation>
        </xss:annotation>
    </xss:element>
    <xss:element name="proxyRef" type="xs:IDREF" minOccurs="0" maxOccurs="1">
        <xss:annotation>
            <xss:documentation source="description">
                Network proxy reference.
            </xss:documentation>
        </xss:annotation>
    </xss:element>
    <xss:element name="clientId" type="xs:string" minOccurs="0">
        <xss:annotation>
            <xss:documentation source="description">
                Client ID that was assigned to this application.
            </xss:documentation>
        </xss:annotation>
    </xss:element>
    <xss:element name="secret" type="xs:string" minOccurs="0">
        <xss:annotation>
            <xss:documentation source="description">
                Client secret that was assigned to this application.
            </xss:documentation>
        </xss:annotation>
    </xss:element>
    <xss:element name="baseUri" type="xs:string" minOccurs="0">
        <xss:annotation>
            <xss:documentation source="description">
                The Sumo Logic endpoints base uri.
            </xss:documentation>
        </xss:annotation>
    </xss:element>
    <xss:element name="collectorUrl" type="xs:string" minOccurs="0">
        <xss:annotation>
            <xss:documentation source="description">
                The Sumo Logic collector URL for sending log events.
            </xss:documentation>
        </xss:annotation>
    </xss:element>
    <xss:element name="dashboardUrl" type="xs:string" minOccurs="0">
        <xss:annotation>
            <xss:documentation source="description">
                The Sumo Logic dashboard URL.
            </xss:documentation>
        </xss:annotation>
    </xss:element>
</xss:sequence>
</xss:complexType>

<xss:complexType name="Subscription">
<xss:annotation>
<xss:documentation source="description">
    Subscription configuration
</xss:documentation>
</xss:annotation>
<xss:sequence>
    <xss:element name="emails" type="EmailListType" minOccurs="0"/>
</xss:sequence>
</xss:complexType>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
<xs:complexType name="EmailListType">
    <xs:annotation>
        <xs:documentation source="description">
            Subscription email list
        </xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="email" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="ReleaseBundlesConfigType">
    <xs:annotation>
        <xs:documentation source="description">
            Release bundles configuration.
        </xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="incompleteCleanupPeriodHours" type="xs:long" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    Define the cleanup period of the incomplete release bundles.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="SignedUrlConfigType">
    <xs:annotation>
        <xs:documentation source="description">
            Signed URL configuration.
        </xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="maxValidForSeconds" type="xs:long" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    The maximum number of seconds a generated signed URL can be valid for.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="EulaType">
    <xs:sequence>
        <xs:element name="accepted" type="xs:boolean" minOccurs="0" maxOccurs="1" default="false">
            <xs:annotation>
                <xs:documentation source="description">
                    Accepted EULA Agreement.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="acceptDate" type="xs:string" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    The date the admin accepted the EULA
                </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="KeyPairType">
    <xs:sequence>
        <xs:element name="key" type="xs:ID">
            <xs:annotation>
                <xs:documentation source="description">
                    Key Pair Unique ID.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="alias" type="xs:string">
            <xs:annotation>
                <xs:documentation source="description">
                    Key Pair alias, to be used as filename by REST API when retrieving the public key.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="vaultType" type="xs:string" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    Key Pair vault type, to determine the pair's storage.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="keyType" type="xs:string">
            <xs:annotation>
                <xs:documentation source="description">
                    Key Pair type, to determine the pair's encryption type.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

# JFrog Artifactory Documentation Displayed in the header

```
</xs:annotation>
</xs:element>
<xs:element name="privateKey" type="xs:string">
  <xs:annotation>
    <xs:documentation source="description">
      Key Pair private key.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="publicKey" type="xs:string">
  <xs:annotation>
    <xs:documentation source="description">
      Key Pair public key.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="passphrase" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      Key Pair passphrase, optional.
    </xs:documentation>
  </xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="RetentionSettingsType">
  <xs:sequence>
    <xs:element name="JFrogColdStorageArchivingNamespace" type="xs:string">
      <xs:annotation>
        <xs:documentation source="description">
          The Archiving Namespace identifier on the JFrog Cold Storage instance
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="RetentionPolicies" type="RetentionPolicyType" maxOccurs="unbounded"
               minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="RetentionPolicyType">
  <xs:sequence>
    <xs:element name="key" type="xs:ID">
      <xs:annotation>
        <xs:documentation source="description">
          Retention Policy Unique ID.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="enabled" type="xs:boolean" default="false">
      <xs:annotation>
        <xs:documentation source="description">
          If enabled, policy will be allowed to execute.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="description" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation source="description">
          Policy Description.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="aqlQuery" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation source="description">
          AQL query for selecting what artifacts to archive.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="cronExp" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation source="description">
          The Cron expression by which retention policy frequency is determined.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="type" type="xs:string">
      <xs:annotation>
        <xs:documentation source="description">
          Type of this policy (ARCHIVE / CLEANUP).
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="durationInMinutes" type="xs:long" default="0" minOccurs="0">
      <xs:annotation>
        <xs:documentation source="description">
          Policy duration in minutes.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="expirationTimeInMonths" type="xs:long" default="0" minOccurs="0">
      <xs:annotation>
        <xs:documentation source="description">
          The expiration period which will be used to calculate the forward expiry date of any artifact
          that will be archived by this Archive Policy
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

# JFrog Artifactory Documentation Displayed in the header

```
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="skipTrashcan" type="xs:boolean" default="false">
<xs:annotation>
<xs:documentation source="description">
    If true, will prevent the transfer of the artifacts to the trashcan repository, and allow the
    artifacts to be
    deleted by a full GC cleanup.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="packageRetention" type="xs:boolean" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
    Explicitly tells if this retention policy is package-oriented
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="searchCriteriaForm" type="RetentionPolicySearchCriteriaType" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
    Search criteria form used to select candidates for artifact archival.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="packageSearchCriteriaForm" type="RetentionPolicyPackageSearchCriteriaType" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
    Search criteria form used to select candidates for package archival.
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="RetentionPolicySearchCriteriaType">
<xs:sequence>
<xs:element name="properties" type="RetentionPolicyPropertiesCriteriaType" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
    List of properties to include/exclude in the search criteria.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="repositories" type="RetentionPolicyPathsCriteriaType" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
    List of repositories to include/exclude in the search criteria.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="paths" type="RetentionPolicyPathsCriteriaType" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
    List of paths to include/exclude in the search criteria.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="names" type="RetentionPolicyPathsCriteriaType" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
    List of names to include/exclude in the search criteria.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="createdBefore" type="xs:long" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
    Relative time (in months)
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="downloadedBefore" type="xs:long" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
    Relative time (in months)
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="RetentionPolicyPackageSearchCriteriaType">
<xs:sequence>
<xs:element name="packageType" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
    A package type (for example "docker")
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="componentName" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation source="description">
    Name of the specific component (package). Can be a wildcard pattern
</xs:annotation>
```

# JFrog Artifactory Documentation

## Displayed in the header

```
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="version" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      Version of the component. Can be a wildcard pattern
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="packageRepos" type="RepositoryListType" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      A list of repository keys to be used for the search
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="createdBefore" type="xs:long" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      Relative time (in months)
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="downloadedBefore" type="xs:long" minOccurs="0">
  <xs:annotation>
    <xs:documentation source="description">
      Relative time (in months)
    </xs:documentation>
  </xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="RetentionPolicyPropertiesCriteriaType">
  <xs:sequence>
    <xs:element name="include" type="RetentionPolicyPropertiesInclusionCriteriaType" minOccurs="0">
      <xs:annotation>
        <xs:documentation source="description">
          List of properties to include in search criteria.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="exclude" type="RetentionPolicyPropertiesInclusionCriteriaType" minOccurs="0">
      <xs:annotation>
        <xs:documentation source="description">
          List of properties to exclude in search criteria.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="RetentionPolicyPropertiesInclusionCriteriaType">
  <xs:sequence>
    <xs:element name="operator" type="xs:string">
      <xs:annotation>
        <xs:documentation source="description">
          Operator ("and"/"or") to determine the logical relations between criteria values.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="values" type="RetentionPolicyPropertyCriteriaListType" minOccurs="0">
      <xs:annotation>
        <xs:documentation source="description">
          List of property elements.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="RetentionPolicyPropertyCriteriaListType">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="property" type="RetentionPolicyPropertyCriteriaType">
      <xs:annotation>
        <xs:documentation source="description">
          Property element, composed of propertyKey and propertyName.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="RetentionPolicyPropertyCriteriaType">
  <xs:sequence>
    <xs:element name="propertyKey" type="xs:string"/>
    <xs:element name="propertyName" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="RetentionPolicyPathsCriteriaType">
  <xs:sequence>
    <xs:element name="include" type="RetentionPolicyPathsInclusionCriteriaType" minOccurs="0">
      <xs:annotation>
        <xs:documentation source="description">
          List of repositories/paths/names to include in search criteria.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
```

# JFrog Artifactory Documentation Displayed in the header

```
</xs:element>
<xs:element name="exclude" type="RetentionPolicyPathsInclusionCriteriaType" minOccurs="0">
    <xs:annotation>
        <xs:documentation source="description">
            List of repositories/paths/names to exclude in search criteria.
        </xs:documentation>
    </xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="RetentionPolicyPathsInclusionCriteriaType">
    <xs:sequence>
        <xs:element name="operator" type="xs:string">
            <xs:annotation>
                <xs:documentation source="description">
                    Operator ("and"/"or") to determine the logical relations between criteria values.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="values" type="RetentionPolicyPathCriteriaListType" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    List of repositories/paths/names.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="RetentionPolicyPathCriteriaListType">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element name="value" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="RepositoryListType">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element name="repo" type="xs:string"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="RetentionMasterTokenType">
    <xs:complexContent>
        <xs:extension base="AuthenticationTokenType">
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

<xs:complexType name="AuthenticationTokenType">
    <xs:sequence>
        <xs:element name="accessToken" type="xs:string" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    Access token from paired instance
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="baseUrl" type="xs:string" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    Base url from pair token
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="exchangeUrl" type="xs:string" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    Exchange url from pair token
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="pairingUrl" type="xs:string" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    Pairing url from pair token - used to refresh the token when expired
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="expirationInMillis" type="xs:long" default="0" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    Master token expiration in millis from paired instance
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="lastRefreshDateMillis" type="xs:long" default="0" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    When master token was refreshed in millis
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="tokenId" type="xs:string" minOccurs="0">
            <xs:annotation>
                <xs:documentation source="description">
                    Token id from paired instance
                </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

```
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="tokenType" type="xs:string" minOccurs="0">
<xs:annotation>
  <xs:documentation source="description">
    Token type from paired instance
  </xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="scope" type="xs:string" minOccurs="0">
<xs:annotation>
  <xs:documentation source="description">
    Token scope from paired instance
  </xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="AuthenticationType">
<xs:annotation>
  <xs:documentation source="description">
    Authentication tokens list used to store master tokens created by pairing a couple of
    artifactory instances, each token can be reused and referenced by one or more sections.
  </xs:documentation>
</xs:annotation>
<xs:sequence>
  <xs:element name="tokens" type="AuthenticationTokenListType" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="AuthenticationTokenListType">
<xs:sequence minOccurs="0" maxOccurs="unbounded">
  <xs:element name="token" type="AuthenticationTokenType"/>
</xs:sequence>
</xs:complexType>

</xs:schema>
```

## 3.9 | WebStart and Jar Signing

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. For more information on Java Web Start, see the [Oracle documentation for Java Web Start](#).

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 instructions on managing signing keys in Artifactory, see [Manage Signing Keys](#)

### WebUI Changes implemented in Artifactory 7.38.x and above

Security is now called **Authentication Providers**. All the relevant text and images on this page have been updated to reflect this change.

#### 3.9.1 | Managing Signing Keys

Signing keys are managed in the **Administration** module under **Authentication Providers | 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](#).

The following topics review how to manage signing keys:

- Generate JAR Signing Keys
- Set Your Keystore and Keys
- Remove a Key Pair
- Configure Virtual Repositories to Sign JARs

##### 3.9.1.1 | Generate 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](#).

##### 3.9.1.2 | Set 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.

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"

Once your keystore is set in Artifactory you may add key pairs under **Add Key-Pair**.

#### 3.9.1.3 | Remove a Key Pair

To remove a key pair, simply select the key pair and click "Remove".

#### 3.9.1.4 | Configure 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.

### 3.10 | RSA Key Pairs

Artifactory enables the management of multiple RSA signing keys allowing you to store and manage the RSA public and private keys that are used to sign and verify the Alpine Linux Index files.

The RSA Key Pair is assigned to local or virtual Alpine Linux Repositories in the Repository Advanced tab of the Alpine Repository.

#### RSA Keys for Alpine Repositories

If you do not configure RSA keys, users have to use the `allow-untrusted` flag. For more information, see Configuring Alpine Package Manager to work with Artifactory.

##### Note

If Vault is configured, follow the steps in Vault.

The following topics provide more information on managing RSA Pairs:

- [Setting Up RSA Keys Pairs](#)
- [Managing RSA Key Pairs](#)
- [RSA Key Pair REST API Commands](#)

#### 3.10.1 | Setting Up RSA Keys Pairs

Step 1: Generate an RSA Key Pair

- Generate an RSA Key Pair. For more information, see [Build a Public and Private RSA Key](#).

##### Note

When generating RSA private keys for use with the Terraform `artifactory_keypair` resource, include the `-traditional` flag during key generation. Use the command:

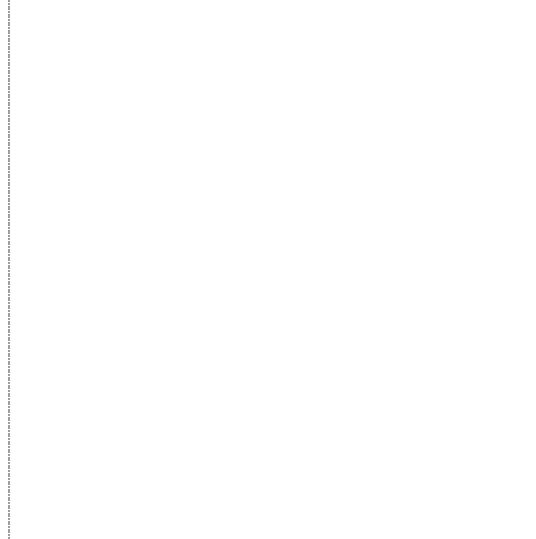
```
openssl genrsa -traditional -out private-key.pem 2048
```

This ensures the private key is properly formatted and prevents the error:

```
Error: RSA private key is of the wrong type.
```

Step 2: Upload the RSA Pair Key

1. In the JFrog Platform UI, go to [Administration module](#) under [Artifactory | Security | RSA Key Pairs](#) and click **Add New Key Pair**.
2. Enter the RSA parameters generated when creating the RSA Key Pair.

- 
3. Click **Save**.

#### 3.10.2 | Managing RSA Key Pairs

In the JFrog Platform, you can view, add or remove the generated RSA Keys in the [Administration module](#), under [Artifactory | Security | RSA Key Pairs](#).

### 3.10.3 | RSA Key Pair REST API Commands

You can perform the following RSA Key Pair REST API commands:

- Create RSA Key Pair
- Get Key Pair
- Delete Key Pair
- Get Key Pair Public Key Per Repository

## 4 | Release Lifecycle Management

An essential part of delivering quality software is creating releases that are validated as they advance through the software development lifecycle (SDLC) toward their eventual consumption by end users. If not managed properly, the lifecycle of releases can become a complex process involving multiple tools and inconsistent processes used by different development teams, leading to an inefficient software supply chain.

JFrog's Release Lifecycle Management solution centers around controlling the flow of a new version of Release Bundles (v2), which are created with the platform UI or with REST APIs from several methods, such as build outputs. The set of artifacts that define a release candidate is wrapped in the Release Bundle, which is signed with its content. The Release Bundle can then be promoted towards production via different stages known as environments (for example, DEV, INT, STG, PROD) and can also be distributed to Distribution Edge nodes.

Users with Artifactory 7.68.9 and above and JFrog Xray 3.82.6 and above can scan the contents of Release Bundles v2 and potentially block these Release Bundles from being promoted if Policy violations are identified. Users with JFrog Distribution 2.20.1 and above can also use Xray to block vulnerable Release Bundles from being distributed.

Each action performed on a Release Bundle is tracked within the JFrog Platform, including creation, promotion, and distribution, and creates signed metadata (known as evidence) attesting to that action. For more information, see [Evidence Management](#).

### Tip

For more details about JFrog's Release Lifecycle Management capabilities and the role they play in ensuring the integrity of your Software Supply Chain, [read this blog](#). You can also submit feedback [here](#).

### Tip

The JFrog CLI includes commands to facilitate the Release Lifecycle Management process. For more information, see [CLI for JFrog Release Lifecycle Management](#).

### Required Subscription Levels

The following subscription levels are required for Release Lifecycle Management operations:

- Create and promote Release Bundles v2: Pro or above
- Scan Release Bundles with Xray: Pro X or above
- Create Xray policies that can control Release Bundle promotion: Enterprise X or above
- Distribute Release Bundles v2 (including distribution in an Air Gap environment): Enterprise+
- Create Xray policies that can control Release Bundle distribution: Enterprise+

### 4.1 | Understanding Release Bundles v2

Release Bundles group together the contents that comprise the bill of materials for your software releases. Each Release Bundle specifies the files and packages that comprise a release, along with their metadata.

Release Bundles exist as a series of one or more versions, each of which must contain at least one artifact. Release Bundles may include source artifacts from local and Federated repositories only. A new Release Bundle can be created in several ways, including from builds, individual artifacts, AQL queries, or existing Release Bundles.

#### Note

Release Bundles cannot contain source artifacts that contain the same path and name from different repositories (known as a path collision).

Since all the artifacts specified in a Release Bundle are required to keep the release coherent, a Release Bundle version is immutable. Effectively, this means that once a Release Bundle version has been created, files cannot be added, modified, or deleted. In addition, any subsequent changes to the properties of the source artifacts are not reflected in the Release Bundle. To change a Release Bundle, a new version must be created.

Each Release Bundle is stored in a Release Bundle repository with a unique name and version per project.

Artifactory tracks all actions related to the Release Bundle version, including its creation, each time it is promoted to a new environment, and when it is distributed. This transparency increases customer confidence in the process that resulted in the software release.

#### 4.1.1 | Types of Release Bundles

The different types of Release Bundles are described below:

##### Release Bundles v1

Many Enterprise+ customers are familiar with the concept of Release Bundles, which are used exclusively by JFrog Distribution to distribute releases to Distribution Edge nodes located at remote sites. This type of Release Bundle is now known as v1.

##### Release Bundles v2

Release Lifecycle Management introduces Release Bundles v2, which can be used as the vehicle within Artifactory for promoting and distributing a release through its lifecycle from the build stage through production.

#### 4.1.2 | Release Bundle v2 Repositories

A Release Bundle v2 is stored as self-contained evidence in a read-only repository with the special package type, `ReleaseBundles`. The Release Bundle repository is created automatically when the first Release Bundle is created.

The Release Bundle specification is stored as a JSON file located under the path `{rb-repository-key}/{rb-name}/{rb-version}` and is signed with a DSSE signature in a Base64-encoded envelope.

In addition to the Release Bundle specification, each Release Bundle contains a snapshot of all included artifacts. Deleting any source artifact from the original path does not compromise the consistency of the contents of the Release Bundle.

The name of the repository used for all Release Bundles created outside the context of a specific project is `release-bundles-v2`. For project-specific Release Bundles, the following prefix is added to the name: `{project-key}-release-bundles-v2`.

#### Note

Any attempt to add a prefix to the Release Bundle repository name that does not match the project name will result in an error.

#### 4.1.3 | Release Bundles v2 and Docker Manifests

A Docker manifest is a JSON file that describes image variants for multiple platforms. If source artifacts contain a Docker manifest (`manifest.json` or `list.manifest.json`), the manifest is resolved automatically during Release Bundle v2 creation. This means the Release Bundle will include both a manifest and all Docker image layers.

To disable this behavior use the following property in the request body:

```
"skip_docker_manifest_resolution": true
```

For example:

```
{
  "release_bundle_name": "Commons-Bundle",
  "release_bundle_version": "1.0.0",
  "skip_docker_manifest_resolution": true,
  "source_type": "aql",
  "source": {
    "aql": "items.find({$and: [{\"repo\": \"$match\": \"repository-prefix-*\"}, {\"name\": \"$match\": \"artifact-prefix-*\"}]}))"
  }
}
```

### 4.2 | Release Lifecycle Management Workflow

# JFrog Artifactory Documentation

## Displayed in the header

The basic workflow for using Artifactory for Release Lifecycle Management is described below.

### One-time Release Bundle v2 Setup by an Administrator

#	Procedure	Using Platform UI	Using APIs
1	Create custom global and project environments, if required	Create a Custom Global Environment Create a Custom Project Environment	Create Global Environment  POST \${baseUrl}/v1/environments  Create Project Environment  POST \${baseUrl}/v1/projects/{project_key}/environments
2	Assign repositories to each environment	Configuring a Local Repository	Update Repository Configuration  Based on changes made in the Repository Configuration JSON.
3	Connect Edge nodes to the JFrog platform  <b>Subscription Information</b>  This feature is supported with the Enterprise+ license.	Connect Distribution Edges to the Platform	Add JPD  POST /mc/api/v1/jpds
4	(optional)  Generate and upload signing keys to Artifactory	Create Signing Keys for Release Bundles (v2)	Create Key Pair  POST /api/security/keypair
5	Propagate signing keys to the Edge nodes  <b>Subscription Information</b>  This feature is supported with the Enterprise+ license.	Not available using the UI. For more information, see:  Propagate Signing Keys to Distribution Edges	Propagate Public Signing Key  POST /lifecycle/api/v2/distribution/key/propagate/{key_name}
6	(optional)  Create webhooks for Release Bundle creation and promotion events	Configure Release Bundle Webhooks  Configure Release Bundle Promotion Webhooks	Create a New Webhook Subscription

### Working with Release Bundles v2

#	Procedure	In Platform UI	Using APIs
1	Create a Release Bundle v2	Create Release Bundles v2	Create Release Bundle Version  POST /lifecycle/api/v2/release_bundle
2	Promote a Release Bundle v2 from one environment to the next	Promote a Release Bundle to a Target Environment	Promote Release Bundle Version  POST /lifecycle/api/v2/promotion/records/{name}/{version}
3	Distribute the Release Bundle v2  <b>Subscription Information</b>  This feature is supported with the Enterprise+ license.	Distribute Release Bundles (v2)	Distribute Release Bundle Version V2  POST /lifecycle/api/v2/distribution/distribute/{release_bundle_name}/{release_bundle_version}

### Tip

The JFrog CLI includes commands to facilitate the Release Lifecycle Management process. For more information, see [CLI for JFrog Release Lifecycle Management](#).

## 4.3 | Release Lifecycle Management Setup

The setup process for Release Lifecycle Management consists of the following steps:

1. Create custom environments (optional)
2. [Enterprise+] Connecting Distribution Edges (optional)
3. Create signing keys
4. [Enterprise+] Propagating signing keys to Edge nodes for distribution
5. Configure Release Bundle webhooks (optional)
6. Configure Release Bundle promotion webhooks (optional)

### Note

For setup instructions when using JFrog Security to scan Release Bundles for vulnerabilities, see [Scan Release Bundles \(v2\) with Xray](#).

### 4.3.1 | Configure Custom Environments

To create and configure custom environments, follow the instructions described in [Manage Custom Environments](#). (If the repository is part of a project, the admin will also need to assign roles to the environment.)

### Note

If the repository is part of a project, the project admin will need to assign roles to the environment.

### 4.3.2 | Connect Distribution Edges to the Platform

To connect Distribution Edges to the JFrog platform, follow the instructions described in [Add an Edge Node](#).

### 4.3.3 | Create Signing Keys for Release Bundles (v2)

When working with Release Bundles v2, you can take the same GPG or RSA signing key added to Artifactory to create and promote the Release Bundle and use it in Distribution to distribute the Release Bundle (Enterprise+ only).

### Important

Starting with Artifactory 7.101.2, Artifactory will create and use a default key (called `default-rsa-key`) if signing keys are not specified during Release Bundle v2 creation.

To generate and upload signing keys for Artifactory (used to create and promote Release Bundles), follow the instructions described in [Generate GPG Keys and Upload GPG Keys](#). Examples are provided below.

#### Generating GPG with RSA key (key size = 2048 bits)

```
# Generate GPG key pair with a passphrase:  
#  
gpg --full-generate-key  
  
# Please select what kind of key you want:  
#  
# > Select "(1) RSA and RSA" by entering "1".  
  
# RSA keys may be between 1024 and 4096 bits long.  
# What keysize do you want? (3072)  
#  
# > Enter "2048"  
  
# Please specify how long the key should be valid.  
#  
# > Enter "0"  
  
# Is this correct? (y/N)  
#  
# > Enter "y"  
  
# GnuPG needs to construct a user ID to identify your key.  
#  
# Real name:  
#  
# > Enter "{your-key-id}"  
#  
# Email address:  
#  
# > Enter "{your-email}"  
#  
# Comment:  
#  
# > Enter "{your-comment}"  
#  
# Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit?  
#  
# > Enter "o"
```

### Note

You may specify a passphrase to use together with the signing keys.

#### Exporting GPG keys

```
# Export the private key with the specified id to a file:  
#  
gpg --output {private-key-output-file} --armor --export-secret-keys {your-key-id}  
  
# Export the public key with the specified id to a file:  
#  
gpg --output {public-key-output-file} --armor --export {your-key-id}
```

#### 4.3.4 | Propagate Signing Keys to Edge Nodes

Use the Propagate Public Signing Key REST API to take the signing key defined in Artifactory for Release Bundles v2 and propagate it to the trusted list of Edge nodes to which the Release Bundles will be distributed.

Using this API eliminates the need to add the key manually to each Edge node one by one.

When working in an Air Gap environment, use this API to add the signing key to the trusted list of the source Artifactory as well. For more information, see [Distribute Release Bundles \(v2\) in an Air Gap Environment](#).

#### 4.3.5 | Configure Release Bundle Webhooks (optional)

Administrators can create webhooks that are triggered by Release Bundle v2 creation events. When a webhook is triggered, it sends relevant information about the event to a web location that is listening for that specific event notification. Webhooks enables you to integrate Release Bundle v2 creation with third-party applications that are also essential to your release lifecycle process.

##### To configure Release Bundle v2 webhooks:

1. In the Administration module, select **General > Webhooks**.
2. Enter a name and description for the webhook.
3. Enter the URL that the webhook should invoke.
4. Under Events, select **Release Bundle V2** and then select one or more of the following events:
  - **Release Bundle creation started**
  - **Release Bundle creation failed**
  - **Release Bundle creation completed**

5. In the Add Release Bundles window, do one of the following:

- Select **Any Release Bundle** to include all Release Bundles in this webhook.
- Choose which Release Bundles to include. Click **Save** when finished.

6. Define the secret token used for authentication.
7. Select the **Use secret for payload signing** checkbox, if required for validating incoming webhook calls.
8. Define custom headers, if required.
9. Click **Create**.

For more information, see [Webhooks](#).

#### 4.3.6 | Configure Release Bundle Promotion Webhooks (optional)

Administrators can create webhooks that are triggered by Release Bundle v2 promotion-related events. When a webhook is triggered, it sends relevant information about the event to a web location that is listening for that specific event notification. Webhooks enables you to integrate Release Bundle v2 promotions with third-party applications that are also essential to your release lifecycle process.

To configure Release Bundle v2 promotion webhooks:

1. In the Administration tab, select **General > Webhooks**.
2. Enter a name and description for the webhook.
3. Enter the URL that the webhook should invoke.
4. Under Events, select **Promotion** and then select one or more of the following events:
  - **Release Bundle promotion started**
  - **Release Bundle promotion failed**
  - **Release Bundle promotion completed**

5. In the Add Environments window, select the environments to which the webhook should apply, and click **Save**.

6. Define the secret token used for authentication.
7. Select the **Use secret for payload signing** checkbox, if required for validating incoming webhook calls.
8. Define custom headers, if required.
9. Click **Create**.

For more information, see [Webhooks](#).

#### 4.4 | Manage the Release Lifecycle Using the Dashboard

The Release Lifecycle dashboard provides an intuitive interface for creating and managing Release Bundles v2. Use the dashboard to manage Release Bundles v2 throughout your release lifecycle, including:

- Creation
- Promotion
- Distribution

# JFrog Artifactory Documentation

## Displayed in the header

### Note

The dashboard requires Artifactory version 7.77.1 or above. Distributing Release Bundles v2 using the dashboard requires Distribution 2.22.1 or above.

#### To open the Release Lifecycle dashboard:

- In the Application module, go to **Artifactory > Release Lifecycle**. The dashboard is displayed.

The dashboard features a table of all existing Release Bundles v2, including the project with which it is associated (if any), the number of versions, and the latest version.

#### Filter the Dashboard

You can filter the information in the Release Lifecycle table in two ways:

- Filter by time:** Use the dropdown list at the top of the window to filter the list of Release Bundles by time (last 7 days, last 30 days, last 90 days, or a custom time period). The default value is **Last 30 Days**.
- Filter by status:** Click the tiles above the table to display all created Release Bundle versions or only those Release Bundle versions that have been promoted to an environment. Enterprise+ users with JFrog Distribution can click the **Distributed** tile to view Release Bundle versions that have been distributed to Edge nodes and other distribution targets.

#### Important

The number of versions shown in the tiles indicates **all** versions within the selected timeframe. However, the table shows only those Release Bundle versions that you have permission to view. Therefore, there might be discrepancies between the number of versions shown in the tiles and the number of versions displayed in the table.

#### Create Release Bundles

Use the **Create Release Bundle** button on the dashboard to create a new Release Bundle v2 version from one or more builds or existing Release Bundles.

#### Additional Views for Managing Release Bundles

The Release Bundle dashboard contains the following additional views for managing Release Bundles:

- Kanban board:** Provides a visual representation of the Release Bundle versions available in each environment defined for your release lifecycle.
- Distribution board:** Provides a visual representation of the Release Bundle versions that have been distributed to Edge nodes and other distribution targets.
- Timeline:** Provides a record of all actions performed on the selected Release Bundle version.

#### 4.4.1 | Use the Release Bundle v2 Promotions Kanban Board

The kanban board provides a visual representation of the versions that exist for the selected Release Bundle and their current location in your release lifecycle. Use the kanban board to promote Release Bundle versions between environments and to access the version **timeline**.

#### To open the promotions kanban board:

- In the Application module, go to **Artifactory > Release Lifecycle**. The dashboard is displayed.
- Click the row for the relevant Release Bundle to display its kanban board.

All Release Bundle versions appear in the Unassigned category after they are created. The card for each version contains the version number, the latest promotion performed on the version, and the date & time on which the version was created/promoted.

As the Release Bundle version is promoted to other environments, its kanban card moves under the relevant category.

In the example shown above, cards for various versions of the Release Bundle named Green Pizza are displayed. As can be seen, each version is located in a different environment, each of which represents a distinct stage in the release lifecycle. In this example, version 4.1.10 has been promoted to the QA environment while versions 4.1.9, 4.1.12, and 4.1.17 have been promoted to PROD. A Release Bundle version can appear in only one environment at a time.

Failed promotions, such as the promotion of version 4.1.15 from DEV to PROD in the example above, are marked with an error banner, as shown below.

Banners are also used to show operations in progress, such as Release Bundle version creation or promotion.

#### Note

Custom environments and the order of their appearance are defined in the Environments window. For more information, see [Create a Custom Global Environment](#) and [Create a Custom Project Environment](#).

#### Kanban Board Actions

You can perform the following actions from the kanban board:

# JFrog Artifactory Documentation

## Displayed in the header

- Promote a Release Bundle version by dragging and dropping it to a different environment. For more information, see [Promote a Release Bundle \(v2\) to a Target Environment](#).
- Click a kanban card to open the timeline for that Release Bundle version. For more information, see [Use the Release Bundle Version Timeline](#).
- Use the Actions menu to create a new version of the selected Release Bundle. For more information, see [Create a New Version of an Existing Release Bundle](#).

### 4.4.2 | Use the Release Bundle v2 Distribution Board

The distribution board provides a visual representation of the Release Bundle versions that have been distributed to Edge nodes and other distribution targets. Use the distribution board to access the [timeline](#) for the relevant Release Bundle version.

#### Subscription Information

This feature is supported with the **Enterprise+** license.

#### Note

Distributing Release Bundles v2 using the dashboard requires Distribution 2.22.1 or above.

#### To open the distribution board:

1. In the Application module, go to [Artifactory > Release Lifecycle](#). The dashboard is displayed.
2. Click the row the relevant Release Bundle.
3. Click the **Distributions** tab.



All Release Bundle versions appear under the target to which they have been distributed. Each card contains the names of the Release Bundle version and the date when it was distributed.

Click the card to view the [timeline](#) for that Release Bundle version.

For more information about distributing Release Bundles v2, see [Distribute a Release Bundle \(v2\) using the Platform UI](#).

#### Distribution Propagation In Federated Environments

#### Note

This feature is relevant for Artifactory 7.84.12 and above and Distribution 2.25.1 and above.

When working with **Federated repositories** as part of a multi-site environment, a Release Bundle version distributed to an Edge node is propagated to the distribution board of any remote JPD in the Federation that uses the same node. Distribution deletions are propagated as well.

# JFrog Artifactory Documentation

## Displayed in the header

Keep in mind the following:

- The name of the Edge node might appear differently on each JPD, depending on how it is configured locally.
- The ability to view distribution events propagated from other Federation members depends on the permissions defined for those Release Bundles.
- Inactive Edge nodes are not displayed.

### 4.4.3 | Use the Release Bundle Version Timeline

The timeline contains a record of all events related to the Release Bundle version listed in reverse chronological order. Use the timeline Actions menu to perform various actions on the Release Bundle version and to view details of a specific event.

To open the Release Bundle version timeline:

- Click a card in the Release Lifecycle kanban board for the relevant Release Bundle version.



The first event always records the creation of the Release Bundle version. New events are added to the timeline as other actions are performed on the version (using either the platform UI or the REST APIs), such as promotion, distribution, export, and deletion. Users with access to evidence management receive timeline events for evidence generated automatically as Release Bundles are created, promoted, and distributed. Events are also added for external evidence attached to the Release Bundle.

#### Tip

You can select a different version of the Release Bundle from the dropdown list above the timeline.

Each timeline event contains a detailed status description and color indicator:

Status	Color	Description
In Progress	Blue	The operation is currently in progress.
Failed	Red	The operation failed.
Success	Green	The operation was successful.
Deleting	Blue	Deletion of the Release Bundle version or promotion is currently in progress.
Deleted	Gray	The Release Bundle version (or version promotion) has been deleted.

Status	Color	Description
Canceled	Yellow	The distribution or export operation was canceled.
Verified		Indicates the evidence file has been verified using the public key created for this purpose. For more information, see <a href="#">Upload the Public Key to Artifactory</a> .

#### Timeline Events in Federated Environments

When working with Federated repositories as part of a multi-site environment, operations performed on a Release Bundle version on one Federation member are propagated to the timelines of the other members. This includes Release Bundle creation, promotion, and distribution.

#### 4.4.4 | Perform Actions on a Release Bundle Version

Use the Actions menu to perform operations on the selected Release Bundle version.

#### In the Kanban Board

When viewing the kanban board for the selected Release Bundle version, use the Actions menu to change the signing key. For more information, see [Change the Signing Key](#).

#### In the Timeline

When viewing the timeline for the selected Release Bundle version, use the Actions menu to do the following:

- Promote the Release Bundle version
- Delete the Release Bundle version
- Add evidence to the Release Bundle version
- Download all evidence attached to the Release Bundle version

# JFrog Artifactory Documentation

## Displayed in the header

Users with JFrog Distribution (requires Enterprise+) can also:

- Distribute the Release Bundle version
- Export the Release Bundle version
- Delete the Release Bundle version from a selected distribution target or multiple targets

### 4.4.5 | View Release Bundle (v2) Version Details

Use the timeline for a Release Bundle version to view detailed information about a selected event related to that version. The information displayed depends on the selected event type, as described below.

#### Release Bundle Version Creation

Click the timeline event for creating a Release Bundle version to view the list of packages that comprise the Release Bundle divided by package type (for example, Docker, Maven, generic, and so on). This information is displayed in the **Content** tab.

Click the row for a package type to display the list of packages belonging to that type.

The row for each package contains two links:

# JFrog Artifactory Documentation

## Displayed in the header

- Click the package name to jump to the packages screen for that specific package version.
- Click the [View artifacts](#) link to jump to the Artifacts tree. The package will be listed in the tree under `release-bundles-v2/[rbv2_name]/[rbv2_version]/artifacts/[package]`. Expand the folder for the package to view the individual artifacts.

### Note

Package types that are made up of individual files (generic, Git LFS, and VCS), will display a table of artifacts instead of packages. Click the artifact name to jump to its entry in the Artifacts tree.

### View the Release Bundle JSON Definition

Click the **Spec** tab to view the JSON file containing the Release Bundle definition. For more information, see [View the Definition of a Release Bundle \(v2\)](#).

### Release Bundle Version Promotion

Click the [timeline](#) event for promoting a Release Bundle version to view a list of promoted artifacts and the target repositories to which they were promoted.

### Note

To delete a promotion, see [Delete a Promotion](#).

### Release Bundle Version Distribution

Click the [timeline](#) event for [distributing](#) a Release Bundle version to view details about the event, including who distributed it, its status and duration, and the list of distributed artifacts.

### Release Bundle Version Export

Click the [timeline](#) event for [exporting](#) a Release Bundle version to view details about the event, including who exported it, when it occurred, and the list of artifacts that were exported.

## 4.5 | Create Release Bundles (v2)

### Subscription Information

This feature is supported on the **Cloud (SaaS)** platform with a **Pro**, **Enterprise X**, or **Enterprise+** license, and on the **Self-Hosted** platform with a **Pro**, **Pro X**, **Enterprise X**, or **Enterprise+** license.

Creating a Release Bundle creates an immutable and signed set of content with a specified version identifier. After the Release Bundle has been created, it can be promoted to different environments with full traceability (see [Promote a Release Bundle \(v2\) to a Target Environment](#)).

Release Bundles v2 are created using one of the following methods:

- Platform UI
- REST API
- JFrog CLI

The platform UI supports the creation of Release Bundle v2 versions from builds and other Release Bundles. In addition to those methods, the REST API and JFrog CLI support Release Bundle v2 creation from AQL and lists of artifacts.

### Important

GPG keys must be created before you can create Release Bundles. See [Create Signing Keys for Release Bundles \(v2\)](#).

#### 4.5.1 | Create a New Release Bundle v2 in the Platform UI

You can use the platform UI to create a Release Bundle (v2) from:

- One or more builds (including aggregated builds)
- Existing Release Bundles (v2)

In addition, you can:

- Create a new version of an existing release bundle
- Create a new release bundle with a different name from an existing build

##### 4.5.1.1 | Create a Release Bundle (v2) from the Builds Table

You can use the Builds table to create a Release Bundle v2 from one or more builds, as an alternative to using the Release Lifecycle dashboard.

The procedures are described in the following sections:

- Create a Release Bundle v2 from One or More Builds
- Create a Release Bundle v2 from a Single Build

### Note

If the Release Bundle is created from a build associated with a project, the Release Bundle will also be associated with the project, even when working in a global context (i.e. when **All** is selected from the project selector).

#### Create a Release Bundle v2 from One or More Builds

Perform the procedure below to create a Release Bundle v2 from one or more builds.

To create a Release Bundle v2 from one or more builds:

1. In the Application module, select **Artifactory > Builds**.

2. Click **Create Release Bundle**.

3. In the **Release Bundle Details** tab of the New Release Bundle window, do the following:

- a. Enter a name for the Release Bundle. (If a Release Bundle was previously created from the selected build, its name will appear.)
- b. Enter the version number of the Release Bundle.

#### Note

Naming restrictions:

- The name is limited to 128 characters.
- The version is limited to 32 characters.
- The name and version must begin with a letter, digit, or underscore.
- The name and version must consist only of letters, digits, underscores, periods, and hyphens.

c. [optional] Select a signing key. If you do not select a key, a default key (generated by Artifactory automatically) is used. See [Create Signing Keys for Release Bundles \(v2\)](#).

#### Warning

Make sure the signing key is valid. The operation will fail if the key has expired.

4. Click **Next** to move to the **Builds Selection** tab.

5. Define the builds from which to create the Release Bundle:

- a. Select the name and version of the build from the lists provided.
- b. [optional] Select the **Include Build Dependencies** checkbox to have the Release Bundle include the dependencies associated with the build.

#### Important

Dependencies must be located in local or Federated repositories to be included in the Release Bundle. Dependencies located in remote repositories are not supported.

To avoid this limitation, copy the dependency artifacts from the remote repository to a local repository before creating the Release Bundle from the build. When you create the Release Bundle from the build (including its dependencies), the dependency artifacts will be resolved with preference to the local repository instead of the remote cache repository.

c. Click **Add**. The selected build and version appear in the selection box.

6. Click **Create**. The Release Bundle appears:

- On the kanban board.
- As an event in the Release Bundle version timeline.
- In the Release Bundle column of the Builds table.

**Note**

To exclude selected artifacts in the build from the Release Bundle, see [Exclude Artifacts from a Release Bundle \(v2\)](#).

**Create a Release Bundle v2 from a Single Build**

Perform the procedure below to create a Release Bundle v2 from a single, selected build.

1. In the Application module, select **Artifactory > Builds**.
2. Click the name of the desired build, and then select **Create Release Bundle** from the actions menu.

**Tip**

Alternatively, click the name of a specific Release Bundle version and then click **Create Release Bundle**.

3. In the **Release Bundle Details** tab of the New Release Bundle window, do the following:
  - a. Enter a name for the Release Bundle. (If a Release Bundle was previously created from the selected build, its name will appear.)
  - b. Enter the version number of the Release Bundle.

### Note

Naming restrictions:

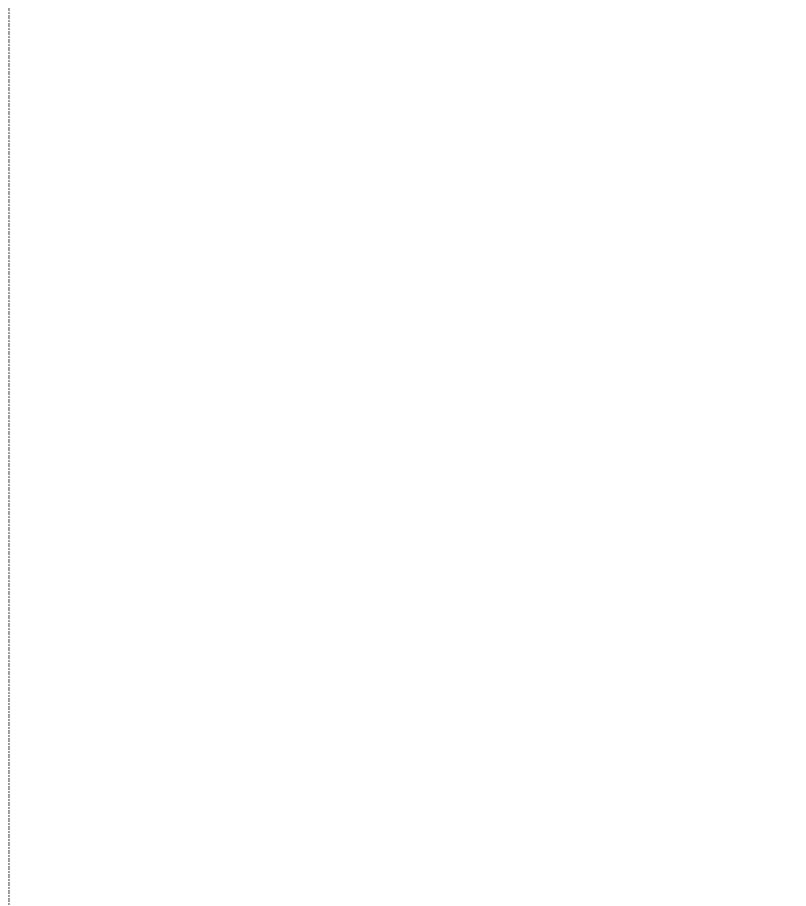
- The name is limited to 128 characters.
- The version is limited to 32 characters.
- The name and version must begin with a letter, digit, or underscore.
- The name and version must consist only of letters, digits, underscores, periods, and hyphens.

c. [optional] Select a signing key. If you do not select a key, a default key (generated by Artifactory automatically) is used. See [Create Signing Keys for Release Bundles \(v2\)](#).

4. Click **Next**.

5. In the **Contents** tab, review the list of modules (containing artifacts and dependencies) to be included in the Release Bundle:

- If the build contains a manifest, the contents of the Release Bundle will be based on the manifest. Using a manifest ensures that only those files that are meant to reach production are included in the Release Bundle. For more information, see [Release Bundle Manifest for Builds](#).
- If the build does not contain a manifest, the entire contents of the build artifacts will be included. This might include supporting files that are used during development but are not part of the final product.



### Note

In the current version, you cannot modify the list of contents in the UI. To modify the contents of the Release Bundle, you need to create a new manifest or use the APIs to create the Release Bundle. The property key of the manifest must be `buildInfo.env.RBV2_MANIFEST`.

6. (optional) Select the **Include Build Dependencies** checkbox to have the Release Bundle include the dependencies associated with the build.

### Important

Dependencies must be located in local or Federated repositories to be included in the Release Bundle. Dependencies located in remote repositories are not supported.

To avoid this limitation, copy the dependency artifacts from the remote repository to a local repository before creating the Release Bundle from the build. When you create the Release Bundle from the build (including its dependencies), the dependency artifacts will be resolved with preference to the local repository instead of the remote cache repository.

7. Click **Create**. The Release Bundle appears:

- On the kanban board.
- As an event in the Release Bundle version [timeline](#).
- In the Release Bundle column of the Builds table.

# JFrog Artifactory Documentation

## Displayed in the header

In addition to creating a Release Bundle v2 from builds, it is possible to create a Release Bundle from one or more existing Release Bundles. For example, this can be used when multiple development teams are working on different microservices, each of which has its own Release Bundle. When it comes time to release a new version of the entire product, the various Release Bundles for the microservices can be combined into a single Release Bundle containing the complete release.

### To create a Release Bundle v2 from existing Release Bundles:

1. In the Application module, select **Artifactory > Release Lifecycle**. The Release Lifecycle dashboard is displayed.
2. Click the **Create Release Bundle** button and then select **From Release Bundles** from the list.



3. In the Release Bundle Details tab of the New Release Bundle window, do the following:

- a. Enter a name for the Release Bundle. (If a Release Bundle was previously created from the selected build, its name will appear.)
- b. Enter the version number of the Release Bundle.
- c. [optional] Select a signing key. If you do not select a key, a default key (generated by Artifactory automatically) is used. See [Create Signing Keys for Release Bundles \(v2\)](#).

### Note

Naming restrictions:

- The name is limited to 128 characters.
- The version is limited to 32 characters.
- The name and version must begin with a letter, digit, or underscore.
- The name and version must consist only of letters, digits, underscores, periods, and hyphens.

### Warning

Make sure the signing key is valid. The operation will fail if the key has expired.

4. Click **Next** to move to the Bundles Selection tab.
5. Define the Release Bundle from which to create the Release Bundle:
  - a. Select the name and version of the Release Bundle from the list provided.
  - b. Click **Add**. The selected Release Bundle and version appear in the selection box.
6. [optional] Repeat step 5 to add additional Release Bundles to the new Release Bundle.

7. Click **Create**.

#### 4.5.1.3 | Create a New Version of an Existing Release Bundle

You can create a new version of an existing Release Bundle v2 from the kanban board.

##### To create a new version of an existing Release Bundle v2:

1. In the Application module, select **Artifactory > Release Lifecycle**. The Release Lifecycle dashboard is displayed.
2. Click the name of the relevant Release Bundle to open the kanban board.
3. From the Actions menu, do one of the following:
  - Select **Create from Release Bundle**.
  - Select **Create from Build**.

4. Leave the name of the Release Bundle as is.

5. Enter a new version number in the field provided.

### Warning

Version number restrictions:

- The version is limited to 32 characters.
- The version must begin with a letter, digit, or underscore.
- The version must consist only of letters, digits, underscores, periods, and hyphens.

6. [optional] Select a new signing key.

7. Click **Confirm**.

### Note

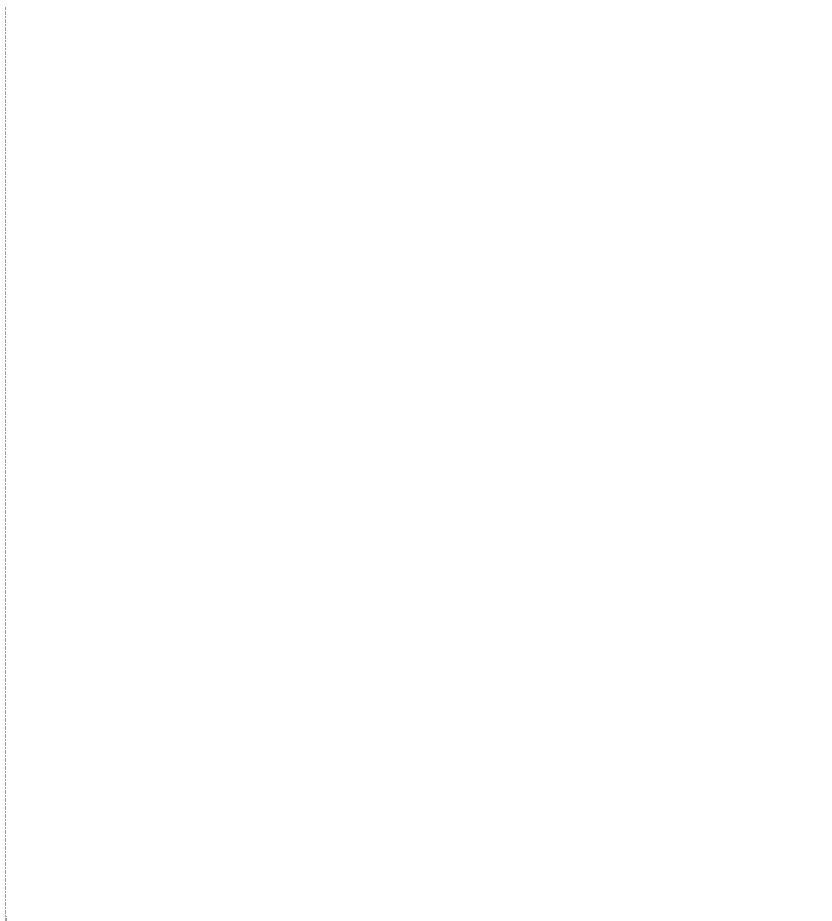
To change the signing key of an existing Release Bundle version, see [Change the Signing Key](#).

### Creating a New Release Bundle Version from the Build Window

If the Release Bundle was created from a single build, you can also return to the build from which the previous version was created and create a Release Bundle with a different version number. By default, the name is the same one that was used for the previous version. (Defining a new name creates a new Release Bundle.) The signing key is predefined with the key that was used for the previous version.

#### 4.5.1.4 | Create a New Release Bundle from the Same Build

To create a new Release Bundle from the same build that was used to create a different Release Bundle, select the **Create new name** checkbox and enter a new name. After entering a unique name for the new Release Bundle, you have the option to select a different signing key than the one used for the previous Release Bundle.



### Note

This option is not available for Release Bundles created from multiple builds or from other Release Bundles.

#### 4.5.1.5 | Exclude Artifacts from a Release Bundle (v2)

When creating a Release Bundle v2 from a single build or multiple builds, you can exclude selected artifacts from the build by defining a Release Bundle manifest for the relevant build.

##### To exclude artifacts from a Release Bundle (v2):

1. Create the Release Bundle manifest JSON file. This is where you define include and exclude patterns for the artifacts inside the build. For examples, see [Release Bundle Manifest for Builds](#).

# JFrog Artifactory Documentation

## Displayed in the header

2. Upload the file to the repository of your choice in Artifactory.
3. Add the Linux local environment variable, RBV2\_MANIFEST, using the following path:

```
export RBV2_MANIFEST="build-manifests-local/<manifest-name>.json"
```
4. Use the following JFrog CLI commands to collect the build info:

```
jfrog rt build-collect-env <build-name> <build-version>
jfrog rt build-publish <build-name> <build-version>
```

The RBV2\_MANIFEST variable becomes a property of the build-info JSON.
5. Unset the RBV2\_MANIFEST variable:

```
unset RBV2_MANIFEST
```
6. Use the API (or UI) to create your Release Bundle v2 from the build. The Release Bundle will include or exclude the artifacts defined in the manifest:

```
POST /lifecycle/api/v2/release_bundle
```

### Note

You can set build parameters, as described in Source Type - Builds.

#### 4.5.2 | Create a New Release Bundle v2 using the REST API

Use the Create Release Bundle v2 Version API to create a new Release Bundle. When using the API, Release Bundles can be created from any of the following source types:

- AQL
- Artifacts
- Builds
- Release Bundles

#### 4.5.3 | Release Bundle v2 Auto-Creation

In addition to the methods you can use to create a Release Bundle v2, Artifactory can automatically create a Release Bundle v2 version whenever you promote a build.

Release Bundles offer several advantages over build promotions. Release Bundles, even if they contain multiple package types, can be promoted with just a couple of clicks, eliminating the need for distinct promotions for each package technology. In addition, Release Bundles provide you with better visibility and control over your release candidate as it progresses through your SDLC. Release Bundles also feature an integrated method to capture evidence of the state of your release candidate and its readiness.

Auto-creation is not dependent on the method used for build promotion, whether you use your CI tool (for example, Bamboo or Jenkins) or the Artifactory REST API.

### Note

Release Bundle v2 auto-creation is controlled by an Artifactory setting for administrators that is enabled by default. For more information, see [Release Lifecycle Settings](#).

### Note

Release Bundle auto-creation is currently supported for Cloud environments only. Support for Self-hosted environments will be added in a later release scheduled for the end of Q1 2025.

### Auto-Creation Guidelines

The following guidelines apply to Release Bundle v2 versions created automatically during build promotion:

- If the build promotion includes dependencies (dependencies=true), the Release Bundle promotion will include dependencies as well.
- The type of build promotion determines the type of Release Bundle promotion:
  - If the build promotion is executed by **moving** the artifacts and dependencies (copy property set to false), the Release Bundle will be promoted to the target environment by moving its artifacts and dependencies.
  - If the build promotion is executed by **copying** the artifacts and dependencies (copy property set to true), the Release Bundle will be promoted to the target environment by copying its artifacts and dependencies.
  - If the build promotion involves a **change of status** without moving or copying the build's contents (artifacts and dependencies both set to false), no Release Bundle is created.
- The Release Bundle name is identical to the build name, except that any special characters (aside from underscore, hyphen, and period) are converted to underscores (\_).
- The Release Bundle version is based on the build number.
- If the length of the build name or build number exceeds the maximum valid values for a Release Bundle name (128 characters) or version (32 characters), build promotion continues but the Release Bundle is not created.
- The value of the created\_by property for the Release Bundle version is \_system\_.
- The environment in which the Release Bundle version is placed is determined by the environment associated with the target repository (targetRepo) of the build promotion.
- If the target repository is not associated with an environment, an environment is created and assigned automatically. The name of the environment will match the status value of the promotion request. If the status is undefined, auto-creation is not performed.

#### 4.5.4 | View the Contents of a Release Bundle (v2)

# JFrog Artifactory Documentation

## Displayed in the header

The Content tab displays all the artifacts that comprise the selected Release Bundle. For example, if you create a Release Bundle from three existing Release Bundles, the Content tab displays a list of all the artifacts in all three Release Bundles.

### To view the contents of a Release Bundle v2:

1. In the Application module, select **Artifactory > Release Lifecycle**. The Release Lifecycle dashboard is displayed.
2. Click the name of the relevant Release Bundle to open the kanban board.
3. Click the card for the relevant Release Bundle version. The Timeline tab is selected by default.
4. Click the **Content** tab. The list of package types that comprise the Release Bundle is displayed.

- 
5. Click the row for a package type to display the list of packages belonging to that type.



The row for each package contains two links:

- Click the package name to jump to the packages screen for that specific package version.
- Click the **View artifacts** link to jump to the Artifacts tree. The package will be listed in the tree under `release-bundles-v2/[rbv2_name]/[rbv2_version]/artifacts/[package]`. Expand the folder for the package to view the individual artifacts.

### Tip

You can also access the contents of a Release Bundle version from the timeline. Scroll down to the start of the timeline and click the creation event for the Release Bundle version. For more information, see [View Release Bundle \(v2\) Version Details](#).

#### 4.5.5 | View the Definition of a Release Bundle (v2)

The Spec tab displays a JSON file containing the definition of the selected Release Bundle.

To view the Release Bundle specs:

1. In the Application module, select **Artifactory > Release Lifecycle**. The Release Lifecycle dashboard is displayed.
2. Click the name of the relevant Release Bundle to open the kanban board.
3. Click the card for the relevant Release Bundle version to open its timeline.
4. Scroll down to the start of the timeline and click the arrow in the entry for Release Bundle version creation.

5. Click the **Spec** tab to see the JSON file containing the definition of the Release Bundle.

#### 4.5.6 | Change the Signing Key

It is possible to change the signing key used to sign an existing Release Bundle, if required. For example, a new key might be required if the original key is compromised or in cases where the responsibility for the Release Bundle moves to a different department or team in your organization.

You can change the key when creating a new version of the existing Release Bundle or when promoting the current version of the existing Release Bundle.

Distribution, however, always uses the key that was used to **create** the Release Bundle version.

##### Note

It is always possible to change the signing key for the Release Bundle using the REST APIs.

### 4.6 | Promote a Release Bundle (v2) to a Target Environment

A promotion operation represents one step as the release advances through the stages of its lifecycle. Promoting a Release Bundle version copies the contents of the Release Bundle to a defined set of target repositories in a defined environment. For example, you can promote to a Staging environment or to Production.

##### Note

The target repositories must be local, Federated, or virtual, provided the virtual repository contains at least one local repository in the relevant environment (or in no environment).

When a Release Bundle is promoted, evidence in the form of a signed and locked file is attached to the Release Bundle that provides a record of the promotion (when, where, and by whom). This evidence provides full traceability and attests to the immutability of the Release Bundle. Each promotion adds another layer of signed evidence that safeguards the immutability of the Release Bundle.

#### To promote a Release Bundle v2 to a target environment:

1. In the Application module, select **Artifactory > Release Lifecycle**. The Release Lifecycle dashboard is displayed.
2. Click the name of the relevant Release Bundle to open the kanban board.
3. Do one of the following:

## JFrog Artifactory Documentation Displayed in the header

- Click the kanban card for the relevant Release Bundle version to open the timeline view. Select **Actions > Promote**.



Or,

- Drag-and-drop the card representing the relevant Release Bundle version to the desired environment.



### Note

It is possible to promote the Release Bundle version to any environment in either direction on the kanban board.

#### 4. In the **Promotion Environment** tab, do the following:

- (optional) Change the signing key, if desired.
- Choose the promotion type: **Copy Artifacts** or **Move Artifacts**

If you choose **Move Artifacts**, the artifacts contained in the Release Bundle are deleted from the repositories in the source environment during promotion.

- Verify the selected target environment.

5. Click **Next**.
6. In the **Target Repositories** tab, review the list of target repositories for each package type. To modify the list, open the relevant dropdown list and clear the checkmark next to each repository to exclude it from the promotion.

**Note**

There must be at least one repository defined for each package type.

7. Click **Promote**.

**Note**

If any artifact in the promotion has a different checksum than the same artifact in the target environment, the promotion stops, and an error message is displayed.

**Note**

You cannot modify or delete a file that belongs to a promoted Release Bundle. You must first delete the promotion or delete the Release Bundle version altogether before the files can be modified or deleted. This restriction protects the immutability of the Release Bundle.

**Warning**

When JFrog Xray is used to scan Release Bundles for potential vulnerabilities and license violations, Xray will block promotion if the Release Bundle violates a Policy that has a blocking action defined. For more information, see [Scan Release Bundles \(v2\) with Xray](#).

**Warning**

Make sure the signing key is valid. The operation will fail if the key has expired.

### Promoting a Release Bundle v2 using the REST API

To promote a Release Bundle v2 version from one environment to another, use the Promote Release Bundle v2 Version REST API.

#### 4.6.1 | Delete a Promotion

When a promotion is deleted, the artifacts included in the promotion are removed from the target repositories, unless the artifacts are still in use by a different Release Bundle. All evidence associated with the promotion is also deleted.

##### To delete a Release Bundle v2 promotion:

1. In the Application module, select **Artifactory > Release Lifecycle**. The Release Lifecycle dashboard is displayed.
2. Click the name of the relevant Release Bundle to open the kanban board.
3. Click the card for the relevant Release Bundle version to open its [Timeline](#).

4. Click the promotion you wish to delete to open a window showing promotion details.

5. Click the trash can icon. In the confirmation window, click **Delete**. All related artifacts are deleted from the target repositories (unless the artifacts are still in use by a different Release Bundle). Any evidence attached to the promotion is also deleted.

In addition, a new entry is added to the timeline that indicates the promotion was deleted. The original entry for the promotion is crossed out, as shown below.

#### Delete a Promotion using the REST API

To delete the promotion of a Release Bundle v2 version, use the Delete Release Bundle v2 Version Promotion REST API.

### Tip

In addition to deleting Release Bundle versions manually as described above, administrators can define a Cleanup policy that removes Release Bundle versions automatically according to defined criteria. For more information, see Create Cleanup Policy - Release Bundle V2.

## 4.7 | Distribute Release Bundles (v2)

### Subscription Information

This feature is supported with the **Enterprise+** license.

### Note

This section describes how to distribute Release Bundles v2. For instructions about distributing Release Bundles v1, see Distribute Release Bundles (v1).

After creating your Release Bundle (and promoting it through the stages of your SDLC, as needed), you can distribute it to the distribution targets (for example, Edge nodes) to which you have distribution privileges.

Distribution can be performed using the JFrog Platform UI, the Distribute Release Bundle Version v2 REST API, or the JFrog CLI.

Distribution is responsible for triggering the replication process that happens from the source Artifactory to the Edge nodes. First, it replicates the Release Bundle information to each Edge node and then initiates the replication process in the source Artifactory. For more details, see The Distribution Flow.

### Prerequisites

- Propagate the signing key defined in Artifactory to the Edge nodes. See [Propagate Signing Keys to Edge Nodes](#).

### Warning

Make sure the signing key is valid. The operation will fail if the key has expired.

- Use one of the following methods to connect Artifactory and the Edge nodes:

- Using Scoped Tokens (Pairing Tokens): Connect the source Artifactory and the edge nodes using the JFrog Platform scoped tokens function. This is the default option used to connect the source and node. (If you cannot upgrade your self-hosted instance, or need to continue using the Circle of Trust, see the next method).
- Circle of Trust: Establish a circle of trust between your source Artifactory and the Edge nodes. This should be applied to both the source and target Artifactory nodes.

### 4.7.1 | Distribute a Release Bundle (v2) using the Platform UI

Release Bundle v2 versions are distributed from the version timeline.

### Subscription Information

This feature is supported with the **Enterprise+** license.

#### To distribute a Release Bundle v2:

1. In the Application module, select **Artifactory > Release Lifecycle**. The Release Lifecycle dashboard is displayed.

### Note

Distributing Release Bundles v2 using the dashboard requires Distribution 2.22.1 or above.

2. Click the name of the relevant Release Bundle to open the kanban board.
3. Click the kanban card for the relevant Release Bundle version to open its timeline.
4. Find the Release Bundle version you wish to distribute and from the Actions menu, select **Distribute**. The Distribute window is displayed.

5. In the Distribution Targets tab, select the desired distribution targets (e.g. Edge nodes).

**Note**

A checkmark next to a grayed-out distribution target indicates that the target already contains the selected Release Bundle version. To redistribute the same Release Bundle version to that target, you must first delete the previous distribution, as described in [Delete a Release Bundle \(v2\) from Multiple Edge Nodes](#).

**Tip**

If no distribution targets are displayed, click the link provided in the window to see suggestions about the possible cause.

6. (optional) Select the **Auto create missing repositories** checkbox to have any missing repositories in the distribution targets created automatically. If this option is not selected and a missing repository is detected, distribution will fail.
7. Click **Next** to move to the Additional Settings tab.
8. (optional) Specify a list of input and output regex mapping pairs that define where the queried artifact is located and where it should be placed. Use this option if the path on the target is different than the source path.

There are several options for defining path mappings:

# JFrog Artifactory Documentation

## Displayed in the header

- You can define custom mapping pairs manually and click **Add**.
- You can click **Use Template** to use one of the provided Path Mapping templates:

Template	Description
Change Repository	All files in a specific repository on the source Artifactory service are mapped to a different repository on the target.
Change Folder	All files are moved to a specific folder in the target.
Rename Folder	All files in a specific folder on the source Artifactory service are mapped to a different folder on the target.

- If the Default Path Mapping by Last Promotion feature is enabled (requires Distribution 2.26.1 or higher), the repositories used for the most recent promotion of this Release Bundle version are provided as the default path mapping. If you do not want this path mapping definition, click the X to remove it.

### Important

See [Path Mapping Guidelines for Release Bundles v2](#) for important instructions and limitations regarding path mapping.

9. Click **Distribute**. The status of the distribution operation is displayed in the version timeline. There will be an entry in the timeline for each distribution target. For example, if you distribute the Release Bundle to 3 Edge nodes, three entries will be added to the timeline (one for each target).

In addition, a new card will appear on the distribution board beneath each distribution target.

### Warning

When JFrog Xray is used to scan Release Bundles for potential vulnerabilities and license violations, Xray will block promotion if the Release Bundle violates a Policy that has a blocking action defined. Scan results are shown in the timeline event (as shown below) and in the **Xray** tab of the dashboard.

For more information, see [Scan Release Bundles \(v2\) with Xray](#).

### Note

If the distribution operation included multiple targets, and one or more targets did not receive the Release Bundle version successfully, users will still be able to download the artifacts from those targets that did receive it successfully.

### Tip

To view distribution details after the operation is complete, click the relevant event in the timeline. The Content tab shows the list of artifacts included in the distribution. The Path Mapping tab records the path mapping that was used between the source and target.

#### 4.7.1.1 | Configure Path Mapping from the Last Promotion

Starting with release 2.26.1, JFrog Distribution includes an option for defining a default path mapping when distributing or exporting a Release Bundle v2 version. This option can be enabled in the Distribution YAML file for all Release Bundle v2 distributions or via API for specific distribution operations, as described below.

When the default path mapping option is enabled, the default destination repositories from the last successful promotion, if one exists, will be displayed in the UI. You can use this mapping if desired, replace it with a mapping of your own, or leave the fields blank. If no such promotion has been performed, the path mapping fields in the UI are left blank, and the destination repositories will match those at the source, as before.

For more information about path mapping, see [Distribute a Release Bundle \(v2\) using the Platform UI](#) and [Export Release Bundles v2 in the Platform UI](#).

##### Enable Default Path Mapping

The option for defining a default path mapping is enabled as follows:

- **YAML file:** A new configuration parameter, `default-path-mapping-by-last-promotion` (`default=false`) has been introduced to support this feature at the system level. For more information, see [Distribution Application Config YAML File](#).
- **REST API:** This feature can be enabled using an optional request parameter in the [Distribute Release Bundle Version v2 API](#). This is useful for enabling the feature for specific requests, facilitating a gradual rollout of this feature.

##### Note

If the optional request parameter is configured in the API (true or false), that setting overrides the configuration parameter in the YAML file.

#### 4.7.1.2 | Path Mapping Guidelines for Release Bundles v2

Observe the guidelines below when defining path mappings for distributed Release Bundle v2 versions to Edge nodes.

# JFrog Artifactory Documentation

## Displayed in the header

When the enhanced distribution engine is not activated or not available (before release 2.28.x)

- You can define path mapping only during the first distribution or export of a Release Bundle version.
- Each distribution target must use the same path mapping.
- Once a path mapping has been defined, it cannot be changed afterward.

When the enhanced distribution engine is available and activated (release 2.28.x or later)

### Note

For more information about the enhanced distribution engine, see Distribution 2.28.1.

- Each distribution operation can have one defined path mapping.
- You can define different path mappings for each distribution target by executing multiple distribution operations.
- The path mapping functionality described above does not apply to exported archives in air-gap environments in the current version.
- Once you define path mappings to a target you cannot define a different set of path mappings for the same Release Bundle version to that target.
- You can use the REST API to redistribute a Release Bundle version to a target that already has that version provided the path mapping has not changed. If the path mapping has changed, you must first delete the Release Bundle version from the target before redistributing it. When using the platform UI, you must delete the Release Bundle version before redistributing it even when the path mapping has not changed.
- After activating the enhanced distribution engine, you must create a new version of Release Bundles that were distributed before activation before they can be redistributed.
- If the enhanced distribution engine is activated and later deactivated, you must create a new version of Release Bundles that were distributed during activation before they can be redistributed.
- Enhanced distribution features are available only when the source Artifactory and all Edge nodes have been upgraded to Artifactory 7.101.x or later and Distribution 2.28.x or later. If you try distributing a Release Bundle version to a mix of upgraded and non-upgraded Edge nodes, the operation will fail.

### 4.7.2 | Distribute a Release Bundle (v2) using REST APIs

Use the following REST APIs to distribute Release Bundles (v2):

- Distribute Release Bundle Version V2: Distributes a Release Bundle V2 version to the designated targets.
- Abort Release Bundle V2 Distribution: Aborts the distribution of a Release Bundle V2 version.
- Get Release Bundle V2 Distributions: Returns a list of distributions for a specified Release Bundle V2 version.
- Delete Release Bundle V2 Distribution: Deletes the Release Bundle V2 version from the distribution targets. (The Release Bundle version is not deleted locally.)

### 4.7.3 | Delete a Release Bundle (v2) Version from a Selected Edge Node

Use the Release Lifecycle timeline to delete a Release Bundle v2 version from a selected distribution target, such as an Edge node.

### Tip

To delete a Release Bundle v2 version from multiple Edge nodes in one operation, see [Delete a Release Bundle \(v2\) from Multiple Edge Nodes](#).

#### To delete a Release Bundle v2 version from a selected Edge node:

1. In the Application module, select **Artifactory > Release Lifecycle**. The Release Lifecycle dashboard is displayed.
2. Click the name of the relevant Release Bundle to open the kanban board.
3. Select the **Distributions** tab.



4. Click any of the cards for the relevant Release Bundle version to open the version timeline. For example, if you want to delete version 1.0.0, click the 1.0.0 card under any of the Edge nodes.
5. Click the distribution operation you wish to delete to open a window showing distribution details.

6. Click the trash can icon. In the confirmation window, click **Delete**. All related artifacts are deleted from the distribution target repositories (unless the artifacts are still in use by a different Release Bundle).

In addition, a new entry is added to the timeline that indicates the distribution was deleted from the target.

#### 4.7.4 | Delete a Release Bundle (v2) from Multiple Edge Nodes

You can delete a distributed Release Bundle version from one or more Edge nodes without deleting it from the source Artifactory that performed the distribution. This operation is known as a Remote Delete.

##### Note

It is also possible to delete a Release Bundle version from a single distribution target using the Distribution details window. For more information, see [Delete a Release Bundle \(v2\) Version from a Selected Edge Node](#).

To perform a remote delete from one or more Edge nodes:

1. In the Application module, select **Artifactory > Release Lifecycle**. The Release Lifecycle dashboard is displayed.
2. Click the name of the relevant Release Bundle to open the kanban board.
3. Select the **Distributions** tab.

4. Click any of the cards for the relevant Release Bundle version to open the version timeline. For example, if you want to delete version 1.0.0, click the 1.0.0 card under any of the Edge nodes.
5. From the Actions menu, select **Delete from Target**.

# JFrog Artifactory Documentation

## Displayed in the header

6. Select the Edge nodes from which the Release Bundle version should be deleted and click **Delete**.

The Release Bundle version is removed from each selected Edge node. An event is added to the version timeline for each node.

### Note

To delete the Release Bundle v2 locally from the source Artifactory, see [Delete a Release Bundle \(v2\) Version](#). In contrast to Release Bundles v1, this operation cannot be performed using JFrog Distribution because all Release Bundles v2 are created and managed by Artifactory.

If the Release Bundle v2 is deleted *locally* (that is, directly) from the Edge node, an event is added to the version timeline of the source Artifactory.

### Perform Remote Delete using the REST API

To delete a distributed Release Bundle v2 version from one or more Edge nodes, use the [Remote Delete Release Bundle v2 Distribution REST API](#).

#### 4.7.5 | Configure Deleted-at-Target Scraping Service

JFrog Distribution includes a service that periodically collects metadata about Release Bundle versions that have been deleted locally from a distribution target (typically an Edge node) and sends the information to the source Artifactory. These local deletions are then added by Artifactory as events in the [timeline](#) of the relevant Release Bundle version.

This service was created because Edge nodes cannot initiate communication on their own with Artifactory instances.

### Note

The target scraping service requires Distribution 2.24.0 or higher *and* Artifactory 7.84.3 or higher. Any Release Bundles that were distributed before both supported versions were installed cannot be scraped and reported by this service.

### Distribution Configuration

The scraping service is configured in the Distribution config YAML, as shown below. There is generally no need to change the default settings.

```
task:  
  deleted-at-target-scraping:  
    batch-size: 200      # number of returned records per request  
    interval-seconds: 900    # interval between successive runs of the delete at target scraping job
```

### Artifactory Configuration

The Artifactory side of the scraping service is configured in the system YAML, as shown below.

```
artifactory:  
  target:  
    bundle:  
      actions:  
        cron: "0 0 2 ? * *"  
        deleteAfterHours: 720
```

Parameter	Description
DEFAULT_CRON_PATTERN	The cron pattern that defines when the source Artifactory informs the distribution target to delete the metadata records of deletion events that have been handled by the scraping service.  The default is “0 0 2 ? * *” (every day at 2:00 am).
DEFAULT_DELETE_ACTIONS_AFTER_HOURS	Determines when to delete leftover metadata that was not deleted by the DEFAULT_CRON_PATTERN.  The default is 720 hours (30 days).

## 4.8 | Distribute Release Bundles (v2) in an Air Gap Environment

### Subscription Information

This feature is supported with the **Enterprise+** license.

### Note

This section describes how to distribute Release Bundles v2 in an Air Gap environment. For instructions about distributing Release Bundles v1, see Distribute Release Bundles (v1) in an Air Gap Environment.

Release Bundle v2 distribution works differently in an Air Gap environment, which is an environment containing two or more JFrog Artifactory instances that have no network connection between them.

To perform distribution, you need to:

- Export the Release Bundle v2 to an archive (.zip file).
- Copy the archive to an external device (such as an external hard drive) and transfer it to the network where the destination Artifactory instance is located.
- Import the archive directly into the destination Artifactory instance.

Exporting and importing Release Bundles v2 can be performed in the platform UI or by using a set of REST API commands. CLI support for Release Bundle v2 distribution is also available.

### Important

Before exporting the Release Bundle, you must propagate the key created in Artifactory (to create and promote the Release Bundle) to the Distribution Edges to which the Release Bundle will be distributed. This is done using an API, as described in [Propagate Signing Keys to Distribution Edges](#).

### 4.8.1 | Export Release Bundles v2 in the Platform UI

The procedure below describes how to export a Release Bundle v2 to a .zip file that can be transferred to a different Artifactory instance in an Air Gap environment.

#### To export a Release Bundle v2:

1. In the Application module, select **Artifactory > Release Lifecycle**. The Release Lifecycle dashboard is displayed.

### Note

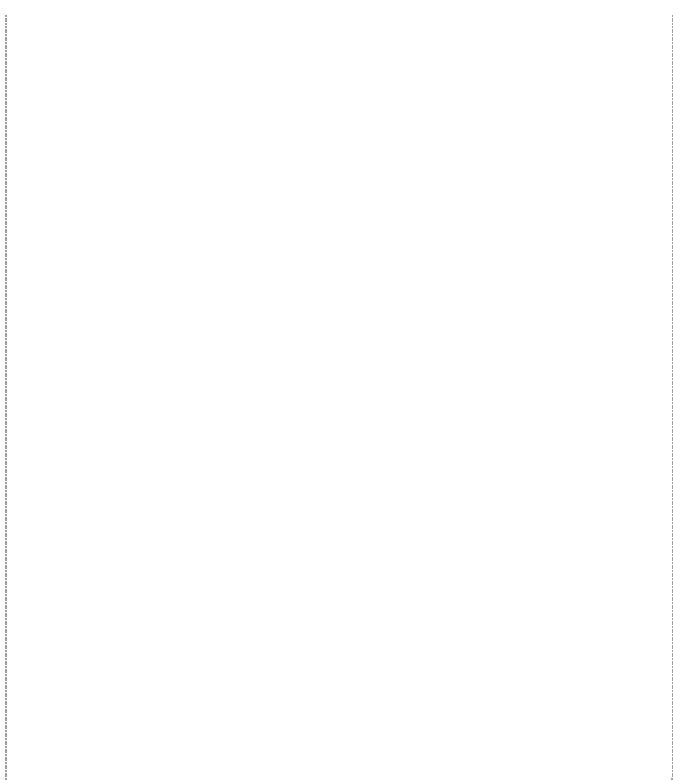
Distributing Release Bundles v2 using the dashboard requires Distribution 2.22.1 or above.

2. Click the name of the relevant Release Bundle to open the kanban board.
3. Click the card for the relevant Release Bundle version to open its timeline.
4. From the Actions menu, select **Export**.

5. (optional) In the Path Mappings field, specify a list of input and output regex mapping pairs that define where the queried artifact is located and where it should be placed after it is imported. Use this option if the path on the target is different than the source path.

### Important

Path mapping can be performed only during the **first** distribution or export of a particular Release Bundle version. The following message is displayed during subsequent operations.



There are 3 options for defining path mappings:

1. You can define custom mapping pairs manually and click **Add**.
2. You can click **Use Template** to use one of the provided Path Mapping templates:

Template	Description
Change Repository	All files in a specific repository on the source Artifactory service are mapped to a different repository on the target.
Change Folder	All files are moved to a specific folder in the target.
Rename Folder	All files in a specific folder on the source Artifactory service are mapped to a different folder on the target.

3. If the Default Path Mapping by Last Promotion feature is enabled (requires Distribution 2.26.1 or higher), the repositories used for the most recent promotion of this Release Bundle version are provided as the default path mapping. If you do not want this path mapping definition, click the X to remove it.

6. Click **Export**. When the process is complete, a new event is added to the version timeline.

#### Warning

When JFrog Xray is used to scan Release Bundles for potential vulnerabilities and license violations, Xray will block the distribution if the License Bundle violates a Policy that has a blocking action defined. For more information, see [Scan Release Bundles \(v2\) with Xray](#).

#### Additional Export Actions

Click the export event in the timeline to view the list of artifacts included in the exported archive. Use the icons to perform the following actions on the archive:

- Delete the exported archive
- Download the exported archive
- Copy the exported archive

##### 4.8.1.1 | [Delete Release Bundle v2 Export](#)

The Export window contains an icon for deleting an archive that you previously exported. This action is typically performed to reclaim storage space.

#### To delete the exported archive of a Release Bundle v2 version:

1. In the Release Bundle version timeline, click the event for the **export** operation.
2. In the Export window, click the **Delete** icon . A confirmation window is displayed.

3. In the confirmation window, click **Delete**. The export archive is deleted. In the timeline, an event is added for the delete operation and the event for the export operation is crossed out.

## JFrog Artifactory Documentation Displayed in the header

### 4.8.1.2 | Download Release Bundle v2 Export Archive

The Export window contains an icon for downloading the exported archive to your local computer. The archive can then be moved manually to another Artifactory instance that is part of an [Air Gap](#) environment.

#### To download an exported Release Bundle v2 archive:

1. In the Release Bundle version timeline, click the event for the [export](#) operation.
  
2. In the Export window, click the **Download** icon . The archive is downloaded to your local computer.

### 4.8.1.3 | Copy Release Bundle v2 Export Link

The Export window contains an icon for copying a link to the exported archive. This makes it possible to share the link with others.

#### To copy the Release Bundle v2 export link:

1. In the Release Bundle version timeline, click the event for the [export](#) operation.
  
2. In the Export window, click the **Copy Link** icon .

### 4.8.2 | Import Release Bundles v2 in the Platform UI

To import the archive containing the exported Release Bundles into the target Artifactory instance, see [Importing a Release Bundle Version](#).

### 4.8.3 | Export and Import Release Bundles v2 using REST APIs

Use the following REST APIs to export Release Bundles v2:

# JFrog Artifactory Documentation

## Displayed in the header

- Export Release Bundle Version v2 Archived File: Exports the Release Bundle to an archive.
- Get Exported Release Bundle Version v2 Status: Returns the status of the export operation.
- Delete Exported Release Bundle v2 Archive File: Deletes an export archive file.

Use the following REST APIs to import Release Bundles v2:

- Import Release Bundle Version: Imports the archive into Artifactory.
- Get Release Bundle Version Import Status: Returns the status of the import operation.

## 4.9 | Download the Contents of a Release Bundle v2

When you are ready to deploy a software release, you can download the contents of a Release Bundle v2 from a target Artifactory, (for example, an Edge node) using the JFrog CLI. For more information, see the [CLI documentation](#).

## 4.10 | Download All Evidence for a Release Bundle Version

You can download a JSON file that contains all the evidence attached to a selected Release Bundle version. The JSON file contains both internal evidence created by Artifactory (for example, during Release Bundle promotion) as well as external evidence added using the JFrog CLI. This feature enables you to save a copy of the evidence locally in a single file.

To download all evidence attached to a Release Bundle version:

1. In the Application module, select **Artifactory > Release Lifecycle**. The Release Lifecycle dashboard is displayed.
2. Click the name of the relevant Release Bundle to open the kanban board.
3. Click the card for the relevant Release Bundle version to open its timeline.
4. From the **Actions** menu, select **Download All Evidence**. A single JSON file containing all internal and external evidence related to the Release Bundle version is downloaded to your local system. The name of the file is <Release\_Bundle\_name>-<version>-Evidence.json.

## 4.11 | Scan Release Bundles (v2) with Xray

### Subscription Information

This feature is supported with **Enterprise X** and **Enterprise+** licenses.

As a Release Bundle v2 is promoted through different environments towards production and eventual distribution to Edge nodes, it is crucial to ensure that the Release Bundle does not contain potential security risks.

JFrog Xray is a universal software composition analysis (SCA) solution that enables you to validate the security risk of a Release Bundle v2 and optionally prevent its promotion and distribution when a risk is detected.

### Note

The ability to scan Release Bundles v2 requires Artifactory version 7.68.6 or later and Xray version 3.82.6 or later.

The workflow for scanning Release Bundles v2 is described below.

**One-time setup procedures performed by a Security Admin:**

- **Step 1 – Index Resources**

For Xray to scan the Release Bundle v2, the Release Bundle must first be added as an indexed resource in Xray. For more information, see [Indexing Xray Resources](#).

- **Step 2 – Create Security Policies**

An Xray administrator can create policies with specific rules, that if violated, can trigger actions including blocking the Release Bundle from being promoted and/or distributed.

For more information about policies, see [Creating Xray Policies and Rules](#).

- **Step 3 – Attach Policies to Watches**

Watches apply a defined policy to specific indexed resources. Use Watches to ensure that Release Bundles are scanned by the appropriate policy. For more information, see [Configuring Xray Watches](#).

**Procedures performed during the software development process:**

# JFrog Artifactory Documentation

## Displayed in the header

### • Step 4 – Scan Release Bundles

Xray scans the Release Bundle as soon as it exists as an indexed resource. Any SCA vulnerabilities discovered in the artifacts that comprise the Release Bundle can be viewed in the Xray tab of the dashboard.

When you act to promote or distribute a Release Bundle, the status of the scan is retrieved from Xray. If Xray detects that the Release Bundle violates one of the rules in the policy defined by the Watch, Xray performs the action defined by the policy. These actions can include blocking the promotion or distribution operation.

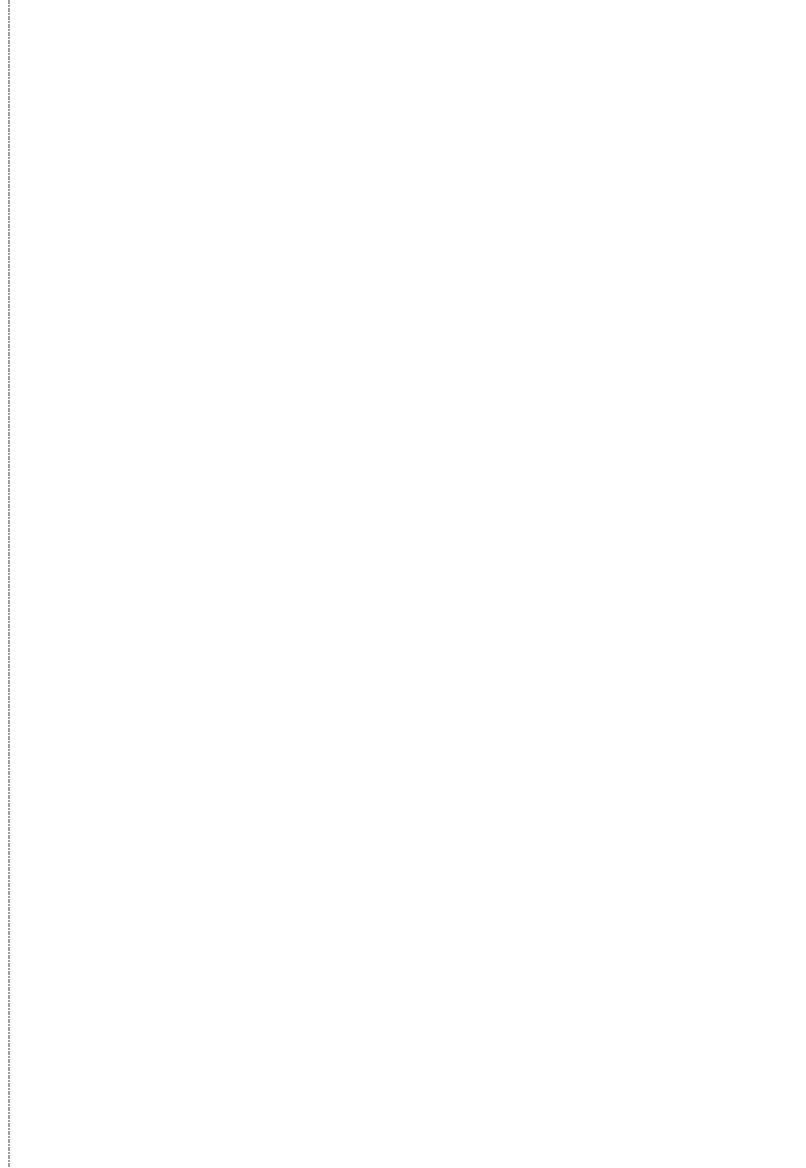
#### Note

The status of the promotion or distribution action remains PROCESSING until the scan is complete.

If the policy blocks the Release Bundle from being promoted or distributed, you can view a list of violations discovered by Xray.

### • Step 5 – View and Evaluate Scan Results

You can view the scan results in the **Xray** tab of the timeline.



The scan results include the number of policy violations detected by Xray and the number of security issues that were uncovered.

#### Tip

You can also collect scan results using REST APIs:

- Scans List - Get Release Bundle v2 Versions: Returns a list of Release Bundle v2 versions that have been scanned by Xray.
- Release Bundle v2 Details: Returns security, license, and operational risk violations found in a specific Release Bundle v2.

### • Step 6 – Resolve Security Issues

Resolve the security issues as necessary to proceed with promotion and distribution. For more information about the different scans provided by Xray and how to resolve issues, see Xray Scan Results.

#### Xray Evidence

Enterprise+ users can view the evidence related to Xray scans during Release Bundle v2 creation, promotion, and distribution. Evidence can be viewed in the evidence graph as well as on the timeline of the Release Bundle version.

# JFrog Artifactory Documentation

## Displayed in the header

When a Release Bundle v2 is created and scanned by Xray, two pieces of evidence are created – an SBOM that lists all the artifacts that comprise the Release Bundle and a second file containing a list of any known vulnerabilities. Both files follow the CycloneDX standard by default.

### Note

To use the SPDX (Software Package Data Exchange) standard instead, change the value of the evidenceSbomFormat property in the Xray system YAML from cyclone to spdx.

### Understanding the Blocking Process

A Release Bundle v2 is blocked by Xray from being promoted and/or distributed when all of the following conditions are met:

- Xray 3.82.6 or higher is installed, enabled, and available.

### Note

There is an optional Xray setting that allows promotion and distribution without scanning if Xray is enabled but unavailable. For more information, see the section, Advanced Settings in Configuring Xray.

- The Release Bundle has been indexed as a resource.
- There is a Watch defined that ties together this Release Bundle with a policy that triggers an action (for example, blocking promotion and/or distribution) if the criteria defined by the policy are met. For example, if a direct vulnerability of a certain severity is discovered in one of the scanned artifacts that comprise the Release Bundle, promotion and distribution of the Release Bundle are blocked.

### Note

In certain circumstances, the indexing process might take a long time to complete. There is a defined timeout setting, which if exceeded, causes the promotion or distribution action to fail if indexing is still in progress or otherwise stuck.

## 4.12 | Monitor Release Bundle v2 Versions

You can gather Release Lifecycle Management performance data using a set of metrics that complies with the Open Metrics standard. This data can then be accessed by 3rd party monitoring tools without the need for custom inbound API calls.

For more information about the available metrics, see Release Lifecycle Management Metrics.

## 4.13 | Verify Release Bundles (v2)

This procedure describes how to verify the signature and contents of a Release Bundle v2. Use this procedure if you want to verify independently that:

- The Release Bundle manifest has not been tampered with en route to its current location (for example, a deployment server)
- The artifacts contained in the Release Bundle have not been tampered with en route

The basic steps are outlined in the following table.

#	Action	Command
1	Get the Release Bundle manifest file	<p>Use the Get Release Bundle v2 Version Signature API:</p> <pre>GET {artifactory-host}/lifecycle/api/v2/release_bundle/signatures/{name}/{version}?project=default</pre> <p>Sample result:</p> <pre>{     "payload":     "eyJfdHlwZSI6Imh0dHBzO18vaW4tdG90by5pb9yTdgf0ZW1lbnQvdjEiLCJzdWJqZWN0IjpbejJ1..."     "payloadType": "application/vnd.in-toto+json",     "signatures": [         {             "keyid": "PGP-RSA-2048",             "sig":             "LS0tLS1CRUdJTiBQR1AgU01HTkFUVVJFLS0tLS0KVmVyc21vbjogQkNQRyB2QFJFTEVBU0..."         }     ] }</pre>
2	Get the public key from Artifactory using the keyid defined in the Release Bundle manifest	<p>Use the Get Key Pair API:</p> <pre>GET {artifactory-host}/artifactory/api/security/keypair/{keyid}</pre> <p>Sample result:</p> <pre>{     "pairName": "PGP-RSA-2048",     "pairType": "PGP",     "alias": "pgp-rsa-2048",     "publicKey": "-----BEGIN PGP PUBLIC KEY BLOCK-----\n\nI0EZJA1AEAA0jFw17MiG608NR60PZ/zpX7Go0jJr2/CNUQwytQXC3BPWMkuVTx\\nxajKzj6...\n-----END PGP PUBLIC KEY BLOCK----",     "unavailable": false }</pre>

# JFrog Artifactory Documentation Displayed in the header

4.14 | Delete a Release Bundle (v2) Version

When a Release Bundle is deleted, it is removed from the Release Bundle repository and all promotion actions associated with the version are removed as well. All evidence associated with the Release Bundle version is also deleted.

### Note

This action does not delete the Release Bundle version from an Edge node. To perform that action, see [Delete a Release Bundle \(v2\) from Multiple Edge Nodes](#).

#### To delete a Release Bundle version:

1. In the Application module, select **Artifactory > Release Lifecycle**. The Release Lifecycle dashboard is displayed.
2. Click the name of the relevant Release Bundle to open the kanban board.
3. Click the card for the relevant Release Bundle version to open its [Timeline](#).
4. From the Actions menu, select **Delete Release Bundle Version**.
5. In the confirmation window, click **Delete**. All promotions associated with the Release Bundle version and their artifacts are deleted. Related evidence is also deleted.

#### Delete a Release Bundle v2 Version using the REST API

To delete a Release Bundle v2 version, use the [Delete Release Bundle v2 Version REST API](#).

### Tip

In addition to deleting Release Bundle versions manually as described above, administrators can define a Cleanup policy that removes Release Bundle versions automatically according to defined criteria. For more information, see [Create Cleanup Policy - Release Bundle V2](#).

## 4.15 | Release Lifecycle Management in Federated Environments

Release Bundles v2 can be used to manage the release lifecycle in a Federated environment, which is where Federated repositories provide Enterprise organizations divided across multiple geographical sites with a single source of truth for their binaries as if they were one seamless unit. This is particularly useful when Federations are employed in a DR (disaster recovery) or Active/Active multi-site framework, as it ensures that your releases, as contained in an immutable Release Bundle, are replicated across all sites.

### Note

The ability to work with Release Bundles v2 in a Federated environment requires Artifactory 7.84.3 or later.

### Prerequisites

The following elements need to be configured on each Federation member:

- **Keys:** Signing keys must be configured on each Federation member.
- **Projects:** Release Bundles v2 created within the scope of a specific project can be Federated only if the project is configured on the other members. If you try to convert a local repository to a Federated repository within the scope of a specific project, and the project is not configured on the other members, the user receives an error.
- **Environments:** Environments and their association with repositories used as promotion targets must be configured on each Federation member.
- **Distribution targets:** Distribution targets such as Edge nodes must be configured on each Federation member. It is possible to configure the same Edge node on each member.
- **Permissions:** Project roles and distribution destinations for each project must be defined for each Federation member.

Once all prerequisites are met, you can manage your SDLC with Release Bundles v2 in a Federated environment.

### Release Bundle v2 Behavior in Federated Environments

The following behavior applies when using Release Lifecycle Management in a Federated environment:

- A Release Bundle v2 version can be created by any member of the Federation. The new version is replicated to all Federation members.
- A Release Bundle v2 version can be promoted and distributed by any member of the Federation (Enterprise+ is required for distribution). However, these operations are not replicated by the other members. The other members receive a timeline entry to inform them of the operation after it is complete.

- 
- If multiple Federation members share the same distribution target (for example, an Edge node), and one member has already distributed a Release Bundle v2 version to that target, a message is displayed ("version already exists") if a second member tries to distribute the same version to the same target.
  - A Release Bundle v2 version can be deleted by any member of the Federation. The deletion is replicated to all Federation members.
  - If multiple Federation members share the same distribution target, deleting a Release Bundle v2 version from the target affects all members that use the target.

### Limitations

Before creating the Federation, verify that the repositories do not contain existing Release Bundles v2 with the same name and version. Creating a Federation under these circumstances can lead to unpredictable behavior, including the possibility of artifacts getting overwritten within those Release Bundles. (Other Release Bundles that do not have clashing names and versions will federate successfully.)

### 4.16 | Known Issues and Limitations

This section describes known issues and limitations related to [Release Lifecycle Management](#).

- Release Bundles v2 are managed in the platform UI under [Application > Artifactory > Release Lifecycle](#). The Release Bundles option found under [Application > Distribution > Release Bundles](#) refers to v1, which is not forward-compatible with v2.
- Xray cannot scan Conan packages inside a Release Bundle v2. As a result, any vulnerabilities and malicious code in those packages will go undetected.
- As of Distribution 2.21.3, Release Bundles v2 created in the context of a specific project can be distributed to Edge nodes and other distribution targets.
- Users with the enhanced distribution engine (Distribution 2.28.1 or higher) should avoid propagating signing keys with identical content to keys saved under a different name on the distribution targets.
- If an Edge node is renamed, the new name will be shown on the [Distribution](#) board, but any distribution events that were added previously to a Release Bundle version [timeline](#) will keep the old name.

## 5 | Evidence Management

### Subscription Information

The ability to collect internal evidence generated by Artifactory requires a **Pro** license or above. Internal evidence generated by Xray requires a **Pro X** license or above.

The ability for users to attach external evidence to Artifactory requires an **Enterprise+** license.

Artifactory enables you to attach evidence (signed metadata) to a designated subject, such as an artifact, build, package, or Release Bundle v2. These evidence files act as attestations, providing a signed and verified record of an external process performed on the subject, for example, test results, vulnerability scans, and official approvals.

JFrog's Evidence service generates an audit trail that documents all the security, quality, and operational steps performed to produce a production-ready software release. It provides a seamless way to consolidate information from the tools and platforms used in software development into a single source of truth that you can track and verify for governance and compliance.

### Note

The Evidence Collection service requires Artifactory release 7.104.2 or later.



You can use any tool to create and sign evidence about a process in your software development lifecycle (SDLC). Collectively, these evidence files can be used to maintain a complete record of your SDLC.

In addition, Artifactory creates internal evidence associated with [Release Lifecycle Management](#) operations, such as Release Bundle v2 promotion and distribution. When integrated with JFrog Xray, each Release Bundle v2 promotion results in the creation of additional evidence, such as scan results and an SBOM.

Evidence management enables you to:

- Attach evidence to an artifact, package, build, or Release Bundle v2 using a REST API or the JFrog CLI.
- View evidence in graphical form and in a table in the [Artifacts tree](#).
- Query evidence via GraphQL APIs. These queries can return the list of evidence files attached to a particular subject and return the contents of a specific evidence file. For more information, see [Search Evidence](#) and [Get Evidence](#).
- Create and apply policies in external applications (for example, Open Policy Agent) that use the information contained in evidence files to block Release Bundle promotions that violate those policies.

### Relationship Between Evidence Files and Subjects

Each evidence subject can have one or more evidence files attached to it. Each evidence file, however, may have only one subject. That subject must reside in a local or Federated repository.

After an evidence file has been attached to a subject (for example, an artifact or Release Bundle v2), the evidence follows the subject as it is promoted towards production. If a subject is copied or moved within Artifactory, its evidence is copied or moved with it.

Users can delete evidence files without affecting the subjects of those files. However, when a subject is deleted from Artifactory, all of its evidence is deleted as well. If the subject is later restored from the Trash Can, the associated evidence is restored. Evidence files cannot be restored from the Trash Can without also restoring their subject.

### To Learn More

- For more information about the ways you can manage evidence in Artifactory, see [Working with Evidence](#).
- For more information about the structure of evidence files, see [Understanding Evidence Files](#).

## 5.1 | Evidence Setup

An administrator should perform the following procedures before you begin uploading evidence to Artifactory:

- Create a key pair for evidence
- Upload the public key to Artifactory

### Create a Key Pair for Evidence

Artifactory supports the following key types for signing and verifying evidence:

- RSA
- EC
- ED25519

The commands for each key type are described in the sections that follow.

### Create an RSA Key Pair

To create an RSA key pair for signing and verifying evidence, issue the following commands on your computer:

```
openssl genrsa -out private.pem 2048
openssl rsa -in private.pem -pubout -out public.pem
```

### Create an EC Key Pair

To create an EC key pair for signing and verifying evidence, issue the following commands on your computer:

```
openssl ecparam -name secp256r1 -genkey -nout -out private.pem
openssl ec -in private.pem -pubout > public.pem
```

### Create an ED25519 Key Pair

To create an ED25519 key pair for signing and verifying evidence, issue the following commands on your computer:

```
openssl genpkey -algorithm ed25519 -out private.pem  
openssl pkey -in private.pem -pubout -out public.pem
```

### Important

It is recommended to use a command line-based copy command, such as `pccopy`, to copy the private key into Artifactory instead of cutting-and-pasting from the terminal UI, which can add stray special characters to the key.

#### Upload the Public Key to Artifactory

After creating the key pair, it is recommended that the administrator upload the public key to Artifactory so that it can be used to verify the evidence on the server. The public key can be uploaded using the platform UI or an API.

##### Upload the Public Key using the Platform UI

For step-by-step instructions, see [Manage Public Keys](#).

##### Upload the Public Key using the REST API

For step-by-step instructions, see [the Set Distribution Public GPG Key](#).

## 5.2 | Working with Evidence

The following tasks can be performed on evidence in the Artifactory platform UI:

- Create external evidence and attach it to an artifact in Artifactory. For more information, see [Attach External Evidence and Create Your Own DSSE](#).
- View evidence files that have been attached to artifacts and Release Bundles v2. For more information, see [View Evidence](#).
- Download all internal and external evidence attached to a Release Bundle v2 version as a single JSON file. For more information, see [Download All Evidence for a Release Bundle Version](#).

The following tasks can be performed on evidence in the REST API:

- Run a Search Evidence GraphQL query to return a list of evidence files associated with a designated subject. For more information, see [Search Evidence](#).
- Run a Get Evidence GraphQL query to return a specific evidence file. For more information, see [Get Evidence](#).

### 5.2.1 | View Evidence

There are several ways to view evidence in Artifactory:

- Select an artifact or build and view a table of evidence files (internal and external) associated with the artifact or build. From the table, you can download evidence files to your local system. For more information, see [View the Artifact Evidence Table](#) and [View the Build Evidence Table](#).
- Select a Release Bundle v2 from the dashboard and view the evidence graph, which provides a graphical depiction of the evidence files associated with the Release Bundle and the builds and packages it contains. For more information, see [View the Release Bundle Evidence Graph](#).
- The evidence graph and the evidence table in the artifact tree can be used to view the [predicate](#), which contains the actual contents of the evidence file. For more information, see [View the Evidence Predicate](#).
- You can view a list of evidence-related events associated with a Release Bundle v2 version (such as promotion to an environment) in the version [timeline](#).

#### 5.2.1.1 | View the Artifact Evidence Table

When you select a Release Bundle v2 version from the [Artifact tree](#), use the **Evidence** tab to view a list of external evidence files associated with the Release Bundle and its contents. You can download these files to your local server.

To view the artifact evidence table:

1. In the **Application** module, select **Artifactory > Artifacts**.
2. Select an artifact from the tree. Details about the artifact are displayed in a series of tabs in the pane to the right of the tree. For more information, see [View Artifact Information](#).
3. Click the **Evidence** tab to display the evidence table.



The table lists all external evidence files related to the selected artifact.

Column	Description
Verified	A blue checkmark icon  indicates the evidence has been verified using the public key created for this purpose. For more information, see <a href="#">Create a Key Pair for Evidence</a> .
Evidence Type	The type of evidence contained in the file, for example, vulnerability scan, code scan, test result, or commit.
Category	The category under which the evidence file is classified, as determined by the predicate type. Categories include: <ul style="list-style-type: none"><li><b>Audit</b> (for example, approvals)</li><li><b>Quality</b> (for example, test results)</li><li><b>Security</b> (for example, evidence related to SAST and code scans)</li><li><b>Workflow</b> (for example, internal evidence relating to the promotion and distribution of Release Bundles)</li><li><b>Custom</b> (other types of external evidence attached by the user)</li></ul>
Time	The timestamp that indicates when the evidence file was created.
Created By	The name of the user who created the evidence file.
Actions menu	Click the icon  to select options for viewing and downloading evidence, as described below.

#### View the Evidence File

To view the contents of the evidence file, select **View Evidence** from the actions menu.

#### Download an Evidence File

To download an evidence file from the table, select **Download Evidence** from the actions menu.

##### 5.2.1.2 | View the Package Evidence Table

When you select a package version from the Packages screen, use the **Evidence** tab to view a list of external evidence files associated with the package. You can download these files to your local server.

###### To view the package evidence table:

1. In the **Application** module, select **Artifactory > Packages**.
2. Click the name of a package to display a table of package versions.
3. Click the relevant version to display the details of that version.
4. Click the **Evidence** tab.
5. Select a repository from the dropdown list above the table.



The table lists all external evidence files related to the package version in the selected repository.

Column	Description
Verified	A blue checkmark icon  indicates the evidence has been verified using the public key created for this purpose. For more information, see <a href="#">Create a Key Pair for Evidence</a> .
Evidence Type	The type of evidence contained in the file, for example, vulnerability scan, code scan, test result, or commit.
Category	The category under which the evidence file is classified, as determined by the predicate type. Categories include: <ul style="list-style-type: none"><li><b>Audit</b> (for example, approvals)</li><li><b>Quality</b> (for example, test results)</li><li><b>Security</b> (for example, evidence related to SAST and code scans)</li><li><b>Workflow</b> (for example, internal evidence relating to the promotion and distribution of Release Bundles)</li><li><b>Custom</b> (other types of external evidence attached by the user)</li></ul>
Time	The timestamp that indicates when the evidence file was created.
Created By	The name of the user who created the evidence file.
Actions menu	Click the icon  to select options for viewing and downloading evidence, as described below.

#### View the Evidence File

To view the contents of the evidence file, select **View Evidence** from the actions menu.

#### Download an Evidence File

To download an evidence file from the table, select **Download Evidence** from the actions menu.

##### 5.2.1.3 | [View the Build Evidence Table](#)

When you select a build version from the Builds screen, use the **Evidence** tab to view a list of external evidence files associated with the build. You can download these files to your local server.

###### To view the build evidence table:

1. In the **Application** module, select **Artifactory > Builds**.
2. Click the name of a build to display a table of build versions.
3. Click the relevant build ID to display the details of that build version.
4. Click the **Evidence** tab to display the evidence table.

The table lists all external evidence files related to the selected artifact.

Column	Description
Verified	A blue checkmark icon  indicates the evidence has been verified using the public key created for this purpose. For more information, see <a href="#">Create a Key Pair for Evidence</a> .
Evidence Type	The type of evidence contained in the file, for example, vulnerability scan, code scan, test result, or commit.
Category	The category under which the evidence file is classified, as determined by the predicate type. Categories include: <ul style="list-style-type: none"><li>• <b>Audit</b> (for example, approvals)</li><li>• <b>Quality</b> (for example, test results)</li><li>• <b>Security</b> (for example, evidence related to SAST and code scans)</li><li>• <b>Workflow</b> (for example, internal evidence relating to the promotion and distribution of Release Bundles)</li><li>• <b>Custom</b> (other types of external evidence attached by the user)</li></ul>
Time	The timestamp that indicates when the evidence file was created.
Created By	The name of the user who created the evidence file.
Actions menu	Click the icon  to select options for viewing and downloading evidence, as described below.

#### View the Evidence File

To view the contents of the evidence file, select **View Evidence** from the actions menu.

#### Download an Evidence File

To download an evidence file from the table, select **Download Evidence** from the actions menu.

##### 5.2.1.4 | View the Release Bundle Evidence Graph

Artifactory uses evidence to provide a complete and verified picture of how a particular release came into being. A way to conceptualize this picture is by using the evidence graph, which consists of evidence artifacts, evidence subjects (in this case, packages), and the relationships between them.

###### To view the evidence graph:

1. In the Application module, select **Artifactory > Release Lifecycle**. The Release Lifecycle dashboard is displayed.
2. Click the name of the relevant Release Bundle to open the **kanban** board.
3. Click the kanban card for the relevant Release Bundle version to open its **timeline**.
4. Click the **Evidence graph** tab to display the evidence graph for this Release Bundle version.

The evidence graph contains the following elements:

Element	Description
	<p>Represents the Release Bundle version, which is the highest element in the graph. The evidence files associated with the Release Bundle appear below it. The builds associated with the Release Bundle appear off to one side with lines connecting them to the Release Bundle.</p>
	<p>Represents a build. The evidence files associated with the build appear below it with lines connecting them to the build.</p> <p><b>Tip</b></p> <p>Click the card to jump to detailed information about the build version.</p>
	<p>Represents a package. The evidence files associated with the artifacts inside the package appear below it with lines connecting them to the package.</p> <p><b>Tip</b></p> <p>Click the card to jump to detailed information about the package.</p>

# JFrog Artifactory Documentation

## Displayed in the header

Element	Description
	<p>Represents an evidence file. Each evidence file is assigned one of the following categories (predicate types):</p> <ul style="list-style-type: none"><li>• Audit Evidence</li><li>• Quality Evidence</li><li>• Security Evidence</li><li>• Workflow Evidence</li><li>• Custom Evidence</li></ul> <p>Click the element for an evidence file to view the predicate, which contains the actual content of the evidence. For more information, see <a href="#">View the Evidence Predicate</a>.</p>
	<p>Indicates the evidence file has been verified using the public key created for this purpose. For more information, see <a href="#">Upload the Public Key to Artifactory</a>.</p>

Click **Show Evidence** to display the evidence cards and **Hide Evidence** to hide them.

### Tip

You can pan the graph left/right and up/down. Use the icons in the corner of the graph to:

Icon	Purpose
	Expand all evidence
	Collapse all evidence
	Zoom in
	Zoom out
	Refresh graph

### 5.2.1.5 | [View the Evidence Predicate](#)

When viewing the evidence graph for a Release Bundle v2 version, click the card for an evidence file to view its predicate, which contains the actual contents of the evidence.

All evidence is available in JSON format. Internal evidence generated by Artifactory (such as Release Bundle v2 promotion reports) and Xray (such as SBOMs and vulnerability reports) is also available in Markdown for easy readability. When both formats are available, the Markdown version is shown in the **Content** tab and the JSON version is shown in the **Spec** tab.

### Tip

The predicate can also be viewed from the Evidence tab in the Artifact tree. For more information, see [View the Artifact Evidence Table](#).

Sample predicates of different types are shown below.

#### Release Bundle v2 Promotion Predicate

Promoting a Release Bundle v2 version creates internal evidence about the event.



#### Release Bundle v2 Distribution Predicate

Distributing a Release Bundle v2 version to a target (such as an Edge node) creates internal evidence about the event.

#### Xray Scan Results

The following are examples of SBOM and vulnerability report evidence generated by Xray in Markdown format.



## Artifact Test Results

Using the [Create Evidence CLI](#), you can attach external evidence to an artifact, such as test results performed outside of Artifactory.

**Tip**

Click the down arrow icon to download the evidence file to your local computer.

**Note**

For more information about attaching external evidence without using the CLI, see [Create Your Own DSSE](#).

### 5.2.2 | Attach External Evidence

#### Subscription Information

This feature is supported with the **Enterprise+** license.

In addition to the evidence that Artifactory and Xray create automatically (for example, when promoting or distributing a Release Bundle v2), you can take external evidence, which attests to processes performed outside of Artifactory, and attach that evidence to an evidence subject (for example, an artifact, package, or build) deployed in Artifactory.

The recommended best practice is to attach evidence to artifacts, packages, or builds until you create a Release Bundle v2 containing those artifacts or builds. At that point, any further evidence related to the artifacts, packages, or builds should be attached directly to the Release Bundle.

One way to attach external evidence is with the [Create Evidence JFrog CLI](#) command. This command creates a properly structured payload that conforms to the [in-toto attestation](#) framework, wraps it in a DSSE envelope, and then deploys the evidence file to Artifactory. Server-side verification is performed automatically during deployment provided the relevant public key is present in Artifactory. For more information, see [Evidence Service](#).

Alternatively, you can create evidence using a third-party tool and then deploy it to Artifactory with the [Deploy Evidence API](#). When using this method, you must ensure that your evidence file has a payload that conforms to the in-toto attestation framework and an envelope that conforms to the DSSE framework.

#### View External Evidence

There are multiple options for viewing external evidence in Artifactory:

- External evidence attached to individual artifacts can be viewed in the [Evidence](#) tab of the Artifacts tree. For more information, see [View the Artifact Evidence Table](#).
- External evidence attached to a Release Bundle v2 can be viewed in the [timeline](#) for that Release Bundle version.
- Evidence attached to a specific Release Bundle version and the artifacts contained in the Release Bundle version can be viewed in the [Evidence graph](#). For more information, see [View the Release Bundle Evidence Graph](#).

### 5.2.3 | Create Your Own DSSE

# JFrog Artifactory Documentation

## Displayed in the header

### Subscription Information

This feature is supported with the **Enterprise+** license.

This section describes how to create your own DSSE envelope containing evidence that can be attached to an artifact in Artifactory without using the Create Evidence CLI. DSSE is a standard for signing arbitrary data, as described [here](#).

### Note

The example below is written in Go, but other languages can also be used.

The complete code can be found at: [https://github.com/jfrog/jfrog-cli-artifactory/blob/main/evidence/create\\_base.go](https://github.com/jfrog/jfrog-cli-artifactory/blob/main/evidence/create_base.go)

1. Load the JSON file (known as the **predicate**) containing the evidence. This is the inner layer of the evidence file.

```
predicate, err := os.ReadFile(ec.predicateFilePath)
if err != nil {
    return err
}
```

2. Next, build the **payload**, which is the middle layer of the evidence file. Create an **in-toto statement** containing the predicate, the predicate type, your server details, and the user.

In addition, you must define the evidence subject, which will be the complete repository path to an artifact already found in Artifactory. You have the option of including the subject's SHA256, which is then validated against the subject in Artifactory.

### Important

In addition to the standard components of an **in-toto statement** (`type`, `predicateType`, and `predicate`), you must also provide the following JFrog extensions:

- `createdAt`  
(must follow the format: `timeLayout = "2006-01-02T15:04:05.000Z"`)
- `createdBy`

```
intotoStatement := intoto.NewStatement(predicate, ec.predicateType, ec.serverDetails.User)
err = intotoStatement.SetSubject(servicesManager, ec.subject)
if err != nil {
    return err
} (edi
```

3. Turn the **in-toto statement** into a JSON file.

```
intotoJson, err := intotoStatement.Marshal()
if err != nil {
    return err
}
```

4. Load your private key. This is the key that will be used to sign the evidence.

```
keyFile, err := os.ReadFile(ec.key)
if err != nil {
    return err
}
```

5. Identify the private key type. (supported types include: RSA, ECDSA, ED25519)

```
privateKey, err := cryptox.ReadKey(keyFile)
if err != nil {
    return err
}
```

6. [optional] Set the key ID. If this option is used, the key ID must match a `keyId` defined in Artifactory.

```
privateKey.KeyID = ec.keyId
```

7. Create a signer based on the private key.

```
signers, err := createSigners(privateKey)
if err != nil {
    return err
}
```

8. Use the signer to create an envelope signer.

```
envelopeSigner, err := dsse.NewEnvelopeSigner(signers...)
if err != nil {
    return err
}
```

9. Sign the DSSE envelope, which is the outer layer of the evidence file. This requires the payload type and the payload created earlier.

```
signedEnvelope, err := envelopeSigner.SignPayload(intoto.PayloadType, intotoJson)
if err != nil {
    return err
}
```

The signed evidence is now ready to be deployed to Artifactory (via the Evidence service).

10. Encode the signed envelope into a byte slice in preparation for deployment (upload) to Artifactory.

```
envelopeBytes, err := json.Marshal(signedEnvelope)
if err != nil {
    return err
}
evidenceManager, err := utils.CreateEvidenceServiceManager(serverDetails, false)
if err != nil {
    return err
}
```

```
}

evidenceDetails := evidenceService.EvidenceDetails{
    SubjectUri: strings.Split(ec.subject, "@")[0],
    DSSEFileRaw: envelopeBytes,
}
_, err = evidenceManager.UploadEvidence(evidenceDetails)
if err != nil {
    return err
}
```

## 5.3 | Understanding Evidence Files

Each piece of evidence can be deployed to Artifactory as a standalone file that is associated with its subject (for example, an artifact or Release Bundle v2). The file name is created automatically.

Evidence files contain multiple layers, as described in the following topics:

- Evidence Predicate
- Evidence Payload
- Evidence Envelope

### 5.3.1 | Evidence Predicate

#### Note

The definition of the predicate is taken directly from the [in-toto attestation framework](#).

The predicate, which is the innermost layer of an evidence file, contains the actual contents of the evidence file. The predicate consists of a JSON file containing arbitrary metadata (one or more claims) about the evidence subject. The user defines the contents and format of the JSON file, but they are typically defined by the file's [predicate type](#).

The predicate is part of the [evidence payload](#), which is the middle layer of an evidence file.

#### Sample Predicate

The following JSON file contains a sample predicate for external evidence about test results performed on an artifact:

```
"tests": {
    "testTool": "Selenium",
    "testStatistics": {
        "passed": 76,
        "failed": 9,
        "other": 6,
        "total": 132
    },
    "tests": [
        {
            "testId": "7878",
            "testName": "Login flow",
            "testDescription": "Tests the login functionality",
            "testStatus": "passed",
            "env": {
                "device": "desktop",
                "os": "windows"
            },
            "steps": [
                {
                    "name": "",
                    "description": "",
                    "expectedResult": ""
                }
            ]
        }
    ]
}
```

### 5.3.2 | Evidence Payload

The payload, which is the middle layer of the evidence file, is based on the [in-toto attestation framework](#), as shown below. The payload contains the [evidence predicate](#), which includes the actual contents of the evidence. The payload is wrapped inside the [Evidence Envelope](#).

```
{
    "_type": "https://in-toto.io/Statement/v1",
    "subject": [
        {
            // Resource Descriptor
        }
    ],
    "predicateType": "{URL-Type}",
    "predicate": {
        // Predicate
    },
    "createdAt": "2022-01-01T00:00:00.000Z",
    "createdBy": "{username}"
}
```

The payload is comprised of the following elements:

# JFrog Artifactory Documentation Displayed in the header

Property	Description
_type	The _type for JFrog evidence is always <a href="https://in-toto.io/Statement/v1">https://in-toto.io/Statement/v1</a> .
subject	Describes the entity associated with the evidence, as defined by its resource descriptor. Each evidence file must be associated with a single subject. For more information, see <a href="#">Evidence Resource Descriptor</a> .
predicateType	<p>Contains the URL type associated with the predicate. The predicate type identifies the meaning of the predicate. Each predicate type is associated with a category to facilitate searching and filtering.</p> <p>Internal predicate types created within Artifactory include:</p> <ul style="list-style-type: none"><li>• <a href="https://jfrog.com/evidence/promotion/v1">https://jfrog.com/evidence/promotion/v1</a></li><li>• <a href="https://jfrog.com/evidence/distribution/v1">https://jfrog.com/evidence/distribution/v1</a></li><li>• <a href="https://cyclonedx.org/bom/v1.4">https://cyclonedx.org/bom/v1.4</a></li><li>• <a href="https://spdx.dev/Document/v2.2">https://spdx.dev/Document/v2.2</a></li></ul> <p>Examples of external predicate types include:</p> <ul style="list-style-type: none"><li>• <a href="https://jfrog.com/evidence/test-results/v1">https://jfrog.com/evidence/test-results/v1</a></li><li>• <a href="https://jfrog.com/evidence/approval/v1">https://jfrog.com/evidence/approval/v1</a></li></ul>
<b>Note</b>	
	For more information about the CycloneDX standard, go <a href="#">here</a> .
	For more information about the SPDX standard, go <a href="#">here</a> .
predicate	Contains the actual body of the evidence file, which is comprised of arbitrary claims about the evidence subject. For more information, see <a href="#">Evidence Predicate</a> .
<b>Note</b>	
	The timestamp when the evidence was created (not when it was deployed to Artifactory).
<b>Note</b>	
	This is a JFrog extension to the standard defined by the in-toto framework.
createdBy	The user who created the evidence.
<b>Note</b>	
	This is a JFrog extension to the standard defined by the in-toto framework.

## Evidence Resource Descriptor

The resource descriptor, which is a mandatory element of the evidence payload, consists of a checksum that represents the evidence subject.

```
{  
    "digest": {  
        "sha256": "ec87961dbf..."  
    }  
}
```

The resource descriptor contains a single element:

Property	Description
digest.sha256	The checksum of the evidence subject.

### 5.3.3 | Evidence Envelope

The outer layer of an evidence file is a **DSSE envelope**, as shown below. It consists of a Base64-encoded payload and a signature.

```
{  
    "payload": "{base64(serialized-Payload)}",  
    "payloadType": "application/vnd.in-toto+json",  
    "signatures": [  
        {  
            "kev-id": "{kev-id}"  
        }  
    ]  
}
```

```
        "sig": "{base64(signature)}"
    }
}
```

The envelope is comprised of the following elements:

Property	Description
payload	Contains a Base64-encoded JSON, as described in Evidence Payload.
payloadType	The payload type for JFrog evidence is application/vnd.in-toto+json.
signatures	The key used to sign the evidence. The array includes the following mandatory elements: <ul style="list-style-type: none"><li>• keyid: Signing key name</li><li>• sig: The signature, as calculated by the DSSE protocol</li></ul>

#### 5.3.4 | Evidence Deployment Workflow

When external evidence is deployed to Artifactory, the evidence file is parsed and validated according to the following sequence:

1. Validates the token used to authenticate the user invoking the Deploy Evidence API
2. Parses and validates the DSSE [evidence envelope](#)
3. Decodes the Base64 payload inside the DSSE envelope
4. Verifies the signature
5. Parses and validates the evidence payload
6. Validates the repository type of the evidence subject (must be local or Federated)
7. Validates the existence of the evidence subject in the specified path in Artifactory
8. Resolves the subject from Artifactory using its full repository path
9. Verifies the subject's digest (if the sha256 is provided in the request)

#### 5.3.5 | Evidence Deletion Workflow

When an evidence subject is deleted from Artifactory, a check is made for any evidence associated with the subject. An asynchronous task deletes the evidence files, and then the evidence subject is deleted.

Evidence can also be deleted explicitly using the Delete Evidence API.

## 6 | Repository Management

This section contains the following topics:

- [Repository Management Overview](#)
- [Local Repositories](#)
- [Remote Repositories](#)
- [Smart Remote Repositories](#)
- [Virtual Repositories](#)
- [Federated Repositories](#)
- [Release Bundle Repositories](#)
- [Repository Replication](#)
- [Repository Layouts](#)
- [Repository Support for Package Clients](#)

### 6.1 | Repository Management Overview

In Artifactory, a repository is a place to organize your artifacts into a cohesive, organized group by application and project.

Artifactory hosts the following repository types:

# JFrog Artifactory Documentation

## Displayed in the header

- Local repositories are physical, locally managed repositories that contain the internal artifacts that originate on your local machine.
- Remote repositories serve as a caching proxy for repositories managed at a remote URL. These repositories contain artifacts that originate outside your local machine, for example, your project's dependencies.
- Virtual repositories aggregate an unlimited number of local and remote repositories to create controlled domains for the search and resolution of artifacts.
- Federated repositories synchronize their contents with other Federated repositories located at remote sites that are part of the same Federation.

### Note

Enterprise+ customers with JFrog Distribution can also create and manage **Release Bundle** repositories, which are used to collect a signed group of artifacts and protect them from changes to ensure consistent distribution to targets, such as JFrog Artifactory Edge nodes.

### Manage Repositories

To create and manage repositories, go to **Repositories** under the **Administration** module.

Administrators can create repositories for a selected package type and assign them to a particular environment. After defining other basic and advanced settings, as required, they can optionally set up replication with a target repository.

### Quick Setup Wizard

You can also use the Quick Setup wizard, which enables you to create repositories for your selected package types in one go. With a couple of simple steps, you can create local, remote, and virtual repositories for each package type of your choosing. For more information, see [Quick Repository Setup](#).

### Uploading Non-Conformant Content

Repositories of each package type have built-in logic for parsing metadata, creating index files, and optimizing performance for packages of that specific type. Uploading non-conformant content, such as images, text files, and other resources that are not wrapped in the proper package format can impact indexing and reduce performance. Therefore, it is strongly recommended to upload generic content to a Generic repository.

#### 6.1.1 | Quick Repository Setup

Use Quick Setup to easily create local, remote, and virtual repositories for your selected package types.

1. Click your login name on the top right-hand corner and select **Quick Repository Creation**.
2. Select one or more package types for which you want to create repositories.

3. Enter a prefix that will be added to the name of each repository (maximum length is 20 characters).

4. Click **Create**.

You have completed creating your repositories, you can continue to configure your clients, and deploy artifacts, as described in [Package Management](#).

#### Learn More

- Best practices for structuring and naming JFrog repositories
- 5 special JFrog repositories you should know about

#### One Package Type Per Repository

When creating a repository, you must select a specific package type; this is a fundamental characteristic of the repository that cannot be changed later. Once the repository type is set, the system 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.

##### Uploading an Incorrect Package Type

While the system 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 the packages you upload. If you upload packages of the wrong type to a repository, Artifactory will not index the package or update the metadata for the repository.

###### 6.1.1.1 | Repository Naming Rules and Limitations

You must adhere to the following rules and limitations when naming repositories:

# JFrog Artifactory Documentation

## Displayed in the header

- Name **cannot** be empty
- Name **cannot** be identical to that of another repository in your Artifactory instance.
- Max number of characters:
  - For Local and Virtual repositories = 64
  - For Remote and Federated repositories = 58
- Name **cannot** contain the following characters: /, \\, :, |, ?, \*, ", <, >
- Name **cannot** consist of the following:
  - .
  - ..
  - &
  - Jfrog-usage-logs
  - Jfrog-billing-logs
  - Jfrog-logs
  - artifactory-build-info
  - artifactory-pipe-info
  - Auto-trashcan
  - jfrog-support-bundle
  - \_intransit
  - Artifactory-edge-uploads
  - release-bundles
- Name **cannot** start with the following:
  - Jfrog-system-reserved
  - Jfrog-artifactory-system
- Name **cannot** end with the following:
  - -cache

For more information, see the whitepaper [Best Practices For Structuring and Naming Artifactory Repositories](#).

### 6.1.1.2 | Generic Repositories

You can define a repository as **Generic** to which you may upload packages of any type. Generic repositories are useful when you want to proxy unsupported package types, store installers, navigation files, audio files, etc.

Since they are not specific to a particular package type, Generic repositories do not maintain separate package indexes. To use a client associated with a specific package type (e.g. yum, gem), you should create a matching repository.

### 6.1.2 | 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, add the repositories to the list of selected repositories in the order in which they should be searched to resolve artifacts.

#### Note

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.

## 6.2 | Local Repositories

Local repositories are physical, locally managed repositories into which you can deploy artifacts. Using local repositories, Artifactory gives you a central location to store your internal binaries. Through repository replication, you can even share binaries with teams that are located in remote locations.

Artifacts in a local repository can be accessed directly using the following URL:

`http://<host>:<port>/artifactory/<local-repository-name>/<artifact-path>`

### 6.2.1 | Configure a Local Repository

The procedure for configuring a local repository includes tabs for basic and advanced settings, as well as a tab for **replication**, if required.

#### To configure a local repository:

1. In the **Administration** module, select **Repositories**.
2. Click **Create a Repository** and select **Local** from the list.

3. In the Select Package Type window, click the icon for the desired package type.

The **Basic** tab for local repositories is displayed.

4. In the **Basic** tab, enter the basic settings for the local repository. For details, see [Basic Settings for Local Repositories](#).
5. In the **Advanced** tab, enter additional advanced settings for the local repository as required. For details, see [Advanced Settings for Local Repositories](#).
6. [optional] In the **Replications** tab, select the checkbox to enable event replication and then add the required replications. For more information, see [Repository Replication](#).
7. When you have finished configuring the repository, click **Create Local Repository**.

#### Configure a Local Repository using the API

Use the **Create Repository** REST API to create a local repository. For more information, see [Create Repository and Repository Configuration JSON](#).

##### 6.2.2 | Basic Settings for Local Repositories

The following basic settings are common for all package types.

Setting	Description
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, unique identifier for the repository. It cannot begin with a number or contain spaces or special characters.
Environments	Defines one or more environments in which this repository will reside. Environments aggregate project resources (repositories, Pipeline sources, etc.) to simplify their management. For more information, see <a href="#">Environments</a> .

# JFrog Artifactory Documentation

## Displayed in the header

Setting	Description
	<p><b>Note</b></p> <p>Defining an environment is mandatory when creating a repository within a specific project. The default selection is DEV.</p>
Repository Layout	Sets the layout that the repository should use for storing and identifying modules. A recommended layout that corresponds to the package type defined is suggested.
Public Description	A free text field that describes the content and purpose of the repository. This description can be viewed by all users with access to the repository.
Internal Description	A free text field to add additional notes about the repository. These notes are visible only to the administrator.
Include and Exclude Patterns	<p>The Include Patterns and Exclude Patterns fields provide a way to filter out specific repositories when resolving 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> <p>Include Patterns: org/apache/**,com/acme/**</p> <p>Exclude Patterns:</p> <p>com/acme/exp-project/**</p> <p>In this example, the repository is searched for org/apache/maven/parent/1/1.pom and com/acme/project-x/core/1.0/nit-1.0.jar but not for com/acme/exp-project/core/1.1/san-1.1.jar because com/acme/exp-project/** is specified as an Exclude pattern.</p>

### Enable Indexing in Xray

JFrog Xray enables repository indexing for security and compliance analysis of the following package types:

Supported Package Types	Unsupported Package Types
Docker	Cargo
npm	Conan
Maven	Conda
PyPI	CRAN
Gradle	Gems
Go	Ivy
Debian	NuGet
RPM	Composer
Terraform BE	SBT
Alpine	Generic
Bower	HuggingFace
	Swift
	Terraform
	Chef
	CocoaPods
	GitLfs
	Helm
	Opkg
	Pub
	Puppet
	Vagrant

### Important

For information about specific settings for particular package types, see [Additional Local Repository Settings for Specific Package Types](#).

#### 6.2.3 | Advanced Settings for Local Repositories

The following advanced settings are common for all package types (with exceptions noted below).

Field	Description
Property Sets	Defines the property sets that will be available for artifacts stored in this repository. For more information, see <a href="#">Property Sets</a> .
Priority Resolution	Setting Priority Resolution takes precedence over the resolution order when resolving virtual repositories. Setting repositories with priority will cause metadata to be merged only from repositories set with this field. If a package is not found in those repositories, Artifactory will merge metadata from the repositories that have not been set with the Priority Resolution field. <b>Note</b> The following package types do not support this field: Cargo, Chef, Debian, Git LFS, Ivy, NuGet, Opkg, Pub, Puppet, Swift, Terraform, Vagrant, and HuggingFace ML
Disable Artifact Resolution in Repository	If set, Artifactory ignores this repository when trying to resolve artifacts. The repository is also not available for download or deployment of artifacts. <b>Note</b> The following package types do not support this field: HuggingFace ML, Debian, Terraform, Terraform BE, Chef, Git LFS, Puppet, Vagrant
Allow Artifact Content Browsing	When set, allows Artifactory users to browse the internal contents of archives (for example, browsing specific Javadoc files from within a Javadoc archive). 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)
Enable CDN Download	Enables CDN Download requests to this repository will redirect the client to download the files directly from AWS CloudFront. Supported for Enterprise+ and Enterprise Licenses. For more information, see <a href="#">JFrog Cloud with CDN Distribution</a> . <b>Note</b> The following package types do not support this field: Alpine, Ansible, Cargo, Chef, CocoaPods, Conda, OCI, Gems, Go, Ivy, NuGet, Opkg, Pub, Puppet, SBT, Swift, Terraform BE, HuggingFace ML, Machine Learning

#### Important

For information about specific settings for particular package types, see [Additional Local Repository Settings for Specific Package Types](#).

#### Replications Tab

The **Replications** tab lets you define and edit replication settings for the repository. For details, refer to [Repository Replication](#).

#### 6.2.4 | Additional Local Repository Settings for Specific Package Types

Local repositories may have additional settings depending on the package type, as described in the following topics:

# JFrog Artifactory Documentation

## Displayed in the header

- Additional Settings for Alpine Local Repositories
- Additional Settings for Cargo Local Repositories
- Additional Settings for CocoaPods Local Repositories
- Additional Settings for Conan Local Repositories
- Additional Settings for Debian Local Repositories
- Additional Settings for Docker Local Repositories
- Additional Settings for Hex Local Repositories
- Additional Settings for Maven/Gradle/Ivy/SBT Local Repositories
- Additional Settings for npm Local Repositories
- Additional Settings for NuGet Local Repositories
- Additional Settings for OCI Local Repositories
- Additional Settings for Opkg Local Repositories
- Additional Settings for PHP Composer Local Repositories
- Additional Settings for RPM Local Repositories
- Additional Settings for Terraform Local Repositories

### 6.2.4.1 | Additional Settings for Alpine Local Repositories

In addition to the basic settings and advanced settings that are common for all local repositories, the following settings are available when configuring Alpine repositories.

Field	Description
Select Key Pair	Select the RSA key pair to use for signing the Alpine Linux index file.

### 6.2.4.2 | Additional Settings for Cargo Local Repositories

In addition to the basic settings and advanced settings that are common for all local repositories, the following settings are available when configuring Cargo repositories.

Field	Description
Allow anonymous download and search	The Cargo client does not send credentials when downloading and searching for crates. When this option is selected, anonymous access is allowed to these resources (only).  <b>Note</b> This option overrides the security anonymous access option.
Enable sparse index support	When selected, this option enables internal index support based on Cargo sparse index specifications, instead of the Git index. For more information, see <a href="#">Index Cargo Repositories Using Sparse Indexing</a> .

### 6.2.4.3 | Additional Settings for CocoaPods Local Repositories

In addition to the basic settings and advanced settings that are common for all local repositories, the following settings are available when configuring CocoaPods repositories.

Field	Description
Custom URL Base	[read-only] Displays the hard-coded URL prefix used to calculate relative URLs.

### 6.2.4.4 | Additional Settings for Conan Local Repositories

In addition to the basic settings and advanced settings that are common for all local repositories, the following settings are available when configuring Conan repositories.

# JFrog Artifactory Documentation

## Displayed in the header

Field	Description
Force Authentication	When selected, this option requires basic authentication credentials to use this repository.

### 6.2.4.5 | Additional Settings for Debian Local Repositories

In addition to the basic settings and advanced settings that are common for all local repositories, the following settings are available when configuring Debian repositories.

Field	Description
Primary Key Name	The name of the GPG primary key used by the repository.
Secondary Key Name	The name of the GPG secondary key used by the repository.
Trivial Layout	When set, the repository will use the deprecated trivial layout.  <b>Important</b> Artifactory no longer supports the trivial layout for virtual Debian repositories.
Enable indexing with debug symbols (.ddeb)	When set, it enables the indexing of debug symbols for more efficient debugging.
Optional Index Compression Formats	Used for selecting the index file formats to create in addition to the default Gzip (.gzip extension) format, which is created for every Debian repository and cannot be disabled.

### 6.2.4.6 | Additional Settings for Docker Local Repositories

In addition to the basic settings and advanced settings that are common for all local repositories, the following settings are available when configuring Docker repositories.

Field	Description
API Version	Select the Docker API version to use, <a href="#">V1</a> or <a href="#">V2</a> .  <b>Note</b> OCI does not support Docker v1.
Max Unique Tags	Specifies the maximum number of unique tags, per repository, that should be stored for a Docker image. If 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. For more information, see <a href="#">Use Max Unique Tags</a> .
Docker Tag Retention	Controls how many overwrites of the same tag are saved in Artifactory. Tag overwriting occurs when you upload a new revision of a tag name that already exists in your repository. For more information, see <a href="#">Use Tag Retention</a> .
Block pushing of image manifest v2 schema 1	When set, Artifactory will block the pushing of Docker images with manifest v2 schema 1 to this repository.

### 6.2.4.7 | Additional Settings for Hex Local Repositories

In addition to the basic settings and advanced settings that are common for all local repositories, the following setting is available when configuring Hex repositories.

Field	Description
Select Key Pair	Select the RSA key pair to sign and encrypt content for secure communication between Artifactory and the Mix client.

# JFrog Artifactory Documentation

## Displayed in the header

### 6.2.4.8 | Additional Settings for Maven/Gradle/Ivy/SBT Local Repositories

In addition to the basic settings and advanced settings that are common for all local repositories, there are settings that are specific to the following package types:

- Maven
- Gradle
- Ivy
- SBT

Field	Description
Checksum Policy	<p>The checksum effectively verifies the integrity of a deployed resource. The <b>Checksum Policy</b> determines how Artifactory behaves when a client checksum for a deployed resource is missing or conflicts with the locally calculated checksum.</p> <p>There are two options:</p> <ul style="list-style-type: none"><li>• <b>Verify against client checksums</b> (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.</li><li>• <b>Trust server generated checksums</b> - 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.</li></ul>
Maven Snapshot Version Behavior	<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> <ul style="list-style-type: none"><li>• <b>Unique</b>: Uses a unique, time-based version number.</li><li>• <b>Nonunique</b>: Uses the default self-overriding naming pattern: <code>artifactID-version-SNAPSHOT.type</code></li><li>• <b>Deployer</b>: Uses the format sent by the deployer as is.</li></ul> <p><b>Deployer parameter option</b></p> <p>Metadata will not be generated when selecting the Deployer option. This option should not be used when setting up replication, since each Artifactory instance will need to generate its metadata locally.</p>
<p><b>Maven 3 Only Supports Unique Snapshots</b></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>	
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	When set, enables you to deploy release artifacts into this repository.
Handle Snapshots	When set, enables you to deploy snapshot artifacts into this repository.
Suppress POM Consistency	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. If there is a conflict between these then Artifactory will reject the deployment. You can disable this behavior by setting this checkbox.

### 6.2.4.9 | Additional Settings for npm Local Repositories

In addition to the basic settings and advanced settings that are common for all local repositories, the following settings are available when configuring npm repositories.

Field	Description
Primary Key Name	The name of the primary key used by the repository. npm repositories use key pairs generated by the elliptic curve implementation of RSA (ECDSA).
Secondary Key Name	The name of the secondary key used by the repository. npm repositories use key pairs generated by the elliptic curve implementation of RSA (ECDSA).

## JFrog Artifactory Documentation Displayed in the header

### 6.2.4.10 | Additional Settings for NuGet Local Repositories

In addition to the basic settings and advanced settings that are common for all local repositories, the following settings are available when configuring NuGet repositories.

Field	Description
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.
Force Authentication	When selected, this option requires basic authentication credentials to use this repository.

### 6.2.4.11 | Additional Settings for OCI Local Repositories

In addition to the basic settings and advanced settings that are common for all local repositories, the following settings are available when configuring OCI repositories.

Field	Description
Max Unique Tags	Specifies the maximum number of unique tags, per repository, that should be stored for a OCI 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. For more information, see <a href="#">Use Max Unique Tags</a> .
OCI Tag Retention	Specifies the number of tags that the JFrog platform will retain when they are overwritten. Leaving the field at 1 (default) means that overwritten tags will not be saved. For more information, see <a href="#">Use Tag Retention</a> .

### 6.2.4.12 | Additional Settings for Opkg Local Repositories

In addition to the basic settings and advanced settings that are common for all local repositories, the following settings are available when configuring Opkg repositories.

Field	Description
Primary Key Name	The name of the GPG primary key used by the repository.
Secondary Key Name	The name of the GPG secondary key used by the repository.

### 6.2.4.13 | Additional Settings for PHP Composer Local Repositories

In addition to the basic settings and advanced settings that are common for all local repositories, the following settings are available when configuring PHP Composer repositories.

Field	Description
Enable Composer V1 Indexing	When selected, uses Composer V1 indexing on the repository. <b>Note</b> From Artifactory 7.24.1, all local PHP repositories are created automatically using PHP Composer V2, which supports faster download times and enhanced performance.

### 6.2.4.14 | Additional Settings for RPM Local Repositories

In addition to the basic settings and advanced settings that are common for all local repositories, the following settings are available when configuring RPM repositories.

Field	Description
Primary Key Name	The name of the GPG primary key used by the repository.

# JFrog Artifactory Documentation

## Displayed in the header

Field	Description
Secondary Key Name	The name of the GPG secondary key used by the repository.
RPM Metadata Folder Depth	<p>Informs Artifactory under which level of directory to search for RPMs and save the repodata 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 RPMs and saves the repodata directory at \$REPO-KEY/repoadata.</p> <p>Using a different depth is useful in cases where generating metadata for a repository separates its artifacts by name, version and architecture. This will allow you to create multiple RPM repositories under the same Artifactory RPM repository.</p> <p>For example:</p> <p>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 Depth to 1 and the repodata directory is saved at REPO_ROOT/ARTIFACT_NAME/repoadata :</p> <pre>REPO_ROOT/\$ARTIFACT_NAME/\$ARTIFACT_VERSION/\$ARCHITECTURE/FILE_NAME - or - rpm-local/foo/1.0/x64/foo-1.0-x64.rpm</pre>
	<p><b>Note</b></p> <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 depth.depth.</p>
Auto Calculate RPM Metadata	When set, RPM metadata calculation is automatically triggered by the actions described by the setting above.
Enable File List Indexing	When set, RPM metadata calculation will also include indexing the filelists.xml metadata file.
RPM Group File Names	A comma-separated list of YUM group files associated with your RPM packages.  Note that at each level (depth), the repodata directory in your repository may contain a different group file name, however each repodata directory may contain only 1 group metadata file (multiple groups should be listed as different tags inside the XML file. For more details, see <a href="#">YUM Documentation</a> ).

### 6.2.4.15 | Additional Settings for Terraform Local Repositories

In addition to the basic settings and advanced settings that are common for all local repositories, the following settings are available when configuring Terraform repositories.

Field	Description
Terraform Registry Type	Select one of the following Terraform repository layouts: <ul style="list-style-type: none"><li>• <b>Module:</b> Applies the <code>terraform-module-default</code> repository layout. For more information, see <a href="#">Use Terraform Module Registries</a>.</li><li>• <b>Provider:</b> Applies the <code>terraform-provider-default</code> repository layout. For more information, see <a href="#">Use Terraform Provider Registries</a>.</li></ul>

## 6.3 | 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 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). Use the following URL to directly access artifacts that are already stored in the cache:

```
http://<host>:<port>/artifactory/<remote-repository-name>-cache/<artifact-path>
```

### 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.

### 6.3.1 | Configure a Remote Repository

The procedure for configuring a remote repository includes a tab of mandatory, basic settings, and additional tabs for advanced settings and replication settings.

**To configure a remote repository:**

1. In the **Administration** module, select **Repositories**.
2. Click **Create a Repository** and select **Remote** from the list.

3. In the Select Package Type window, click the icon for the desired package type.

The **Basic** tab for remote repositories is displayed.

4. In the **Basic** tab, enter the basic settings for the remote repository. For details, see [Basic Settings for Remote Repositories](#).
5. [optional] In the **Advanced** tab, enter the advanced settings for the remote repository. For details, see [Advanced Settings](#).
6. [optional] In the **Replications** tab, select the checkbox to enable event replication and then add the required replications. For more information, see [Repository Replication](#).
7. When you have finished configuring the repository, click [Create Remote Repository](#).

# JFrog Artifactory Documentation

## Displayed in the header

### Configure a Remote Repository using the API

Use the [Create Repository](#) REST API to create a remote repository. For more information, see Create Repository and Repository Configuration JSON.

#### 6.3.2 | Basic Settings for Remote Repositories

The following basic settings are common for all package types.

Setting	Description
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, unique identifier for the repository. It cannot begin with a number or contain spaces or special characters.
Environments	Defines one or more environments in which this repository will reside. Environments aggregate project resources (repositories, Pipeline sources, etc.) to simplify their management. For more information, see Environments.

# JFrog Artifactory Documentation

## Displayed in the header

Setting	Description
	<p><b>Note</b></p> <p>Defining an environment is mandatory when creating a repository within a specific project. The default selection is DEV.</p>
URL	The URL for the remote repository. Currently, only HTTP and HTTPS URLs are supported.
User Name	The user name for accessing the URL of the remote repository. Leave this field blank to access the repository anonymously.
Password/Access Token	The password or access token used to access the repository.
SSL/TLS Certificate	The certificate used for authentication. Click <b>Test</b> to verify the certificate is valid.
Repository Layout	Defines the layout used by the remote repository cache for storing and identifying modules. For more information, see <a href="#">Remote Repository Layout Configuration</a> .
Remote Layout Mapping	Defines the layout used by the remote repository for storing and identifying modules. This layout can differ from the layout defined under Repository Layout for the remote repository cache. For more information, see <a href="#">Remote Repository Layout Configuration</a> .
Public Description	A free text field that describes the content and purpose of the repository. This description can be viewed by all users with access to the repository.
Internal Description	A free text field to add additional notes about the repository. These notes are visible only to the administrator.
Include and Exclude Patterns	<p>The Include Patterns and Exclude Patterns fields provide a way to filter out specific repositories when resolving 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> <p>Include Patterns: org/apache/**, com/acme/**</p> <p>Exclude Patterns:</p> <p>com/acme/exp-project/**</p> <p>In this example, the repository is searched for org/apache/maven/parent/1/1.pom and com/acme/project-x/core/1.0/nit-1.0.jar but not for com/acme/exp-project/core/1.1/san-1.1.jar because com/acme/exp-project/** is specified as an Exclude pattern.</p>
Offline	If set, no attempts will be made to fetch remote artifacts from this repository. Only locally-cached artifacts are retrieved.
Enable Indexing in Xray	Enables indexing on the repository for security and compliance analysis. Available with JFrog Xray.

### Enable Indexing in Xray

JFrog Xray enables repository indexing for security and compliance analysis of the following package types:

Supported Package Types		Unsupported Package Types
Docker	Conan	Swift
npm	Conda	Terraform
Maven	CRAN	Chef
PyPI	Gems	CocoaPods
Gradle	Ivy	GitLfs
Go	NuGet	Helm
Debian	Composer	Opkg
RPM	SBT	P2
Alpine	Generic	Pub
Bower	HuggingFace	Puppet
Cargo		VCS

## Important

For information about specific settings for particular package types, see [Additional Remote Repository Settings for Specific Package Types](#).

### 6.3.3 | Advanced Settings for Remote Repositories

The advanced settings for a remote repository configure network access behavior, cache management, and several other parameters related to remote repository access. For more information, see:

- [Remote Credentials](#)
- [Network Settings for Remote Repositories](#)
- [Cache Settings for Remote Repositories](#)
- [Zapping Caches](#)
- [Select Property Sets](#)
- [Other Advanced Settings for Remote Repositories](#)

To access the advanced settings, select the **Advanced** tab when editing an existing **Remote Repository** or creating a new one.

#### 6.3.3.1 | Remote Credentials

You can configure authentication for remote repositories by specifying remote credentials. You can use either one of the following methods to identify yourself:

- [Username and password](#)
- [Personal Access Token \(PAT\)](#)

The advantage of using PATs is that you can strengthen your Artifactory security practices by using them for authentication, instead of using your primary credentials. For example, you can create a PAT in GitHub and then configure your remote Docker repository to point to GitHub and authenticate it by using the PAT. You can use PATs for any package type.

##### To specify remote credentials:

1. In the **Administration** module, select **Repositories**.
2. Click the **Remote** tab and open the relevant remote repository.
3. In the **Basic** tab, enter your remote credentials in the following fields:

4. Use one of the following methods to authenticate yourself when accessing this remote proxy:

a. User name and password:

- i. Under **Remote Authentication**, enter a user name in the **User Name** field.
- ii. Enter a password in the **Password/Access Token** field.

b. Personal Access Tokens (PAT) used for HTTP authentication (Artifactory 6.18 and later):

- i. Create the PAT in the service provider and copy the PAT to the clipboard.
- ii. In Artifactory, paste the PAT into the **Password/Access Token** field.
- iii. Enter a username in the **Username** field. When a PAT is used for authentication, an arbitrary username can be entered into this field (not applicable for `npmjs`), but the field must not be left empty.

**Note**

To access the npm registry, you must pass a valid npm registry username and generated PAT token. For more details, see this [Knowledge Base article](#).

5. Enter the SSL/TLS certificate used for authentication to the remote resource for which this repository is a proxy.

**Tip**

Click **Test** to verify the certificate.

6. Click **Save**.

6.3.3.2 | Network Settings for Remote Repositories

Network settings for remote repositories are described below.

# JFrog Artifactory Documentation

## Displayed in the header

Field	Description
Proxy	If your organization requires you to go through a proxy to access a remote repository, this parameter lets you select the corresponding <b>Proxy Key</b> . For more details on setting up proxies in Artifactory please refer to Managing Proxies.
No Proxy	To prevent auto-updates during the edit of system proxies, a flag was added in release 7.41.7 called <b>No Proxy</b> (in the UI) or <code>disableProxy</code> (in the REST API). The flag is set to <code>false</code> by default - since it turns off the use of a proxy for this repository and prevents proxy updating during system proxy changes. <b>Disabling the Proxy</b> With the release of 7.41.7, the functionality to disable a remote proxy requires you to set this action in the UI (or API) to ensure that no proxy is used.
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 (in ms) 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 <b>Socket Timeout</b> Artifactory registers the repository as "assumed offline" for the period of time specified in Assumed Offline Period.
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&amp;param2=value2&amp;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**.

#### 6.3.3.3 | Cache Settings for Remote Repositories

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.

Field	Description
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 to <a href="#">Cleanup Unused Cached Artifacts</a> in Regular Maintenance Operations</p> <p>Leaving the field empty (default) means that the artifact is stored indefinitely.</p>
Metadata Retrieval Cache Period (sec)	<p>Defines how long before Artifactory checks for a newer version of a requested artifact in a remote repository.</p> <p>A value of 0 means that Artifactory will always check for a newer version.</p> <p><b>On which file types does this parameter work?</b></p> <p>This setting refers to artifacts that expire after a period of time (e.g. metadata files such as <code>maven-metadata.xml</code>, <code>npm package.json</code> or <code>Docker manifest.json</code> etc.).</p> <p>Note that most artifacts that are downloaded do not change (e.g. release versions), therefore this setting does not affect them.</p>
Metadata Retrieval Cache Timeout	<p>Allows you to control the Metadata timeout performance. If the timeout is reached, the previous metadata is returned to the client, as a lock was not applied due to new metadata, leaving the previous request hanging. The default value is <b>60</b> seconds.</p>
Assumed Offline Period	<p>In case of a connection error, this parameter specifies how long (in seconds) Artifactory should wait before attempting an online check to reset the offline status.</p> <p>A value of 0 means that the repository is never assumed offline and Artifactory will always attempt to make the connection when demanded. The default value is <b>300</b> seconds.</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.</p> <p>A value of 0 means that the response is not cached and Artifactory will always issue a new request when demanded. The default value is <b>1800</b> seconds.</p>

#### 6.3.3.4 | Zapping Caches

Zapping cache invalidates all cached metadata artifacts downloaded from central registries like `pypi.org`, `repo1.maven.org`, `registry.npmjs.org` and stored in remote repo cache to speed up remote repo actions.

# JFrog Artifactory Documentation

## Displayed in the header

### Note

The zapping action does not invalidate immutable artifacts (like software binaries). As they are immutable, there is no need to bring them again from the central repository.

After Zapping, whenever an invalidated metadata artifact is needed, Artifactory refreshes it from the central registry first. If unavailable, it falls back to the stale version.

Zapping the cache may slow down for clients who download packages requiring stale metadata updates.

### Zapping cache solves:

- **Resolving Cache Issues:** If there are problems with the cached packages artifacts, such as corruption or inconsistencies with the central repository, zapping the cache fixes these issues.

### Note

Requires **Manage** or **Delete** permissions on the Remote Repository.

For zapping a cache via API, refer to Zap Cache REST API.

### To zap a cache using the platform UI:

1. In the **Artifacts** module tree browser, select the repository cache you wish to "zap".
2. Click **Zap caches** in the right-click menu or from the **Actions** drop-down menu.

### 6.3.3.5 | Select Property Sets

Defines the property sets that will be available for artifacts stored in this repository. For more information, see [Property Sets](#).

### 6.3.3.6 | Other Advanced Settings for Remote Repositories

# JFrog Artifactory Documentation

## Displayed in the header

The following advanced settings are common for most package types.

Field	Description
Priority Resolution	<p>Setting Priority Resolution takes precedence over the resolution order when resolving virtual repositories. Setting repositories with priority will cause metadata to be merged only from repositories set with this field. If a package is not found in those repositories, Artifactory will merge metadata from the repositories that have not been set with the Priority Resolution field.</p> <p>Applies to all repository types excluding Chef, CocoaPods, Debian, Git LFS, Opkg, Rust, Vagrant, and VCS repositories.</p> <p>For more information on package support for this feature, see the <a href="#">Artifactory Release Notes</a>.</p>
Disable Artifact Resolution in Repository	If set, Artifactory ignores this repository when trying to resolve artifacts. The repository is also not available for download or deployment of artifacts.
Allow Artifact 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> <p><b>Warning</b></p> <p>When Allow Content Browsing is enabled, this functionality restricts access to authenticated users only and is not supported for trial users. This limitation exists to prevent malicious users from uploading content that may compromise security.</p>
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> <p>This setting is relevant for all package types, except Ansible, Helm OCI, OCI, and Hugging Face.</p> <p><b>When might you use direct repository-to-client streaming?</b></p> <p>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>
Synchronize Properties	When set, synchronizes properties of artifacts retrieved from a remote instance of Artifactory.
Bypass HEAD Requests	<p>When set, Artifactory will not send a HEAD request to the remote resource before downloading an artifact for caching.</p> <p><b>Note</b></p> <p>CocoaPods and VCS packages do not support this setting.</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 <code>system.properties</code> file under <code>blockedMismatchingMimeTypes</code>. The MIME type identifies the format and content of artifacts in the repository.</p> <p>You can override this setting by adding MIME types to the override list below.</p> <p><b>Note</b></p> <p>This setting can be overridden by the settings of the upstream server. For example, if the upstream server allows all MIME types (Accept: <code>*/*</code>), the upstream server setting will override this Artifactory setting.</p>
Enable CDN Download	Enables CDN Download requests to this repository will redirect the client to download the files directly from AWS CloudFront. Supported for Enterprise+ and Enterprise Licenses. For more information, see <a href="#">JFrog Cloud with CDN Distribution</a> .
Disable URL Normalization	<p><b>Note</b></p> <p>The following package types <b>do not</b> support this setting: Go, Swift, Alpine, Cargo, Chef, CocoaPods, Conda, Gems, Ivy, NuGet, Opkg, P2, Pub, Puppet, SBT, VCS</p>
Retrieve SHA256 from Remove Server	When set, Artifactory retrieves the SHA256 from the remote server if it is not cached in the remote repository.
Override Default Blocked Mime Types	The set of mime types that should override the <b>Block Mismatching Mime Types</b> setting.

### Important

For information about specific settings for particular package types, see [Additional Remote Repository Settings for Specific Package Types](#).

# JFrog Artifactory Documentation

## Displayed in the header

### 6.3.4 | Additional Remote Repository Settings for Specific Package Types

Remote repositories may have additional basic settings depending on the package type, as described in the following topics:

- [Additional Settings for Bower Remote Repositories](#)
- [Additional Settings for Cargo Remote Repositories](#)
- [Additional Settings for CocoaPods Remote Repositories](#)
- [Additional Settings for Composer Remote Repositories](#)
- [Additional Settings for Conan Remote Repositories](#)
- [Additional Settings for Debian Remote Repositories](#)
- [Additional Settings for Docker Remote Repositories](#)
- [Additional Settings for Generic Remote Repositories](#)
- [Additional Settings for Go Remote Repositories](#)
- [Additional Settings for Legacy Helm Remote Repositories](#)
- [Additional Settings for Helm OCI Remote Repositories](#)
- [Additional Settings for Hex Remote Repositories](#)
- [Additional Settings for Maven/Gradle/Ivy/SBT Remote Repositories](#)
- [Additional Settings for NuGet Remote Repositories](#)
- [Additional Settings for Opkg Remote Repositories](#)
- [Additional Settings for PyPI Remote Repositories](#)
- [Additional Settings for RPM Remote Repositories](#)
- [Additional Settings for Swift Remote Repositories](#)
- [Additional Settings for Terraform Remote Repositories](#)
- [Additional Settings for VCS Remote Repositories](#)

#### 6.3.4.1 | Additional Settings for Bower Remote Repositories

In addition to the [basic settings](#) and [advanced settings](#) that are common for all remote repositories, the following settings are available when configuring Bower repositories.

Field	Description
Git providers	Select the Git provider to be used by the remote repository: <ul style="list-style-type: none"><li>• GitHub</li><li>• BitBucket</li><li>• Stash/Private BitBucket</li><li>• Stash/Private BitBucket (prior to 5.1.0)</li><li>• Artifactory (used when configuring a Smart Remote repository to another instance of Artifactory)</li><li>• Custom</li></ul>
Registry URL	Defines the base URL of the registry API used by the remote repository.

#### 6.3.4.2 | Additional Settings for Cargo Remote Repositories

In addition to the [basic settings](#) and [advanced settings](#) that are common for all remote repositories, the following settings are available when configuring Cargo repositories.

Field	Description
Registry URL	Defines the base URL of the registry API used by the remote repository.
Allow anonymous download and search	When selected, the Cargo client does not send credentials when performing download and search for crates. Enable this option to allow anonymous access only to these resources.
Enable sparse index support	When selected, enables internal index support based on Cargo sparse index specifications, instead of the default git index.

## JFrog Artifactory Documentation Displayed in the header

### 6.3.4.3 | Additional Settings for CocoaPods Remote Repositories

In addition to the basic settings and advanced settings that are common for all remote repositories, the following settings are available when configuring CocoaPods repositories.

Field	Description
Git provider	Select the Git provider to be used by the remote repository: <ul style="list-style-type: none"><li>• GitHub</li><li>• BitBucket</li><li>• Stash/Private BitBucket</li><li>• Stash/Private BitBucket (prior to 5.1.0)</li><li>• Artifactory (used when configuring a Smart Remote repository to another instance of Artifactory)</li><li>• Custom</li></ul>
Specs Repo URL	Defines the URL of the public CocoaPods Specs repository. This is a git repository containing podspec.json files pointing from a package name and version to its storage endpoint. It does not contain any actual binary packages.  The default value is <a href="https://github.com/CocoaPods/Specs">https://github.com/CocoaPods/Specs</a> .
CDN URL	Defines the URL of the CocoaPods CDN, which expedites the workflow by creating a static copy of the CocoaPods Specs repository, reducing the time required for adding repositories.  The default is <a href="https://cdn.cocoapods.org">https://cdn.cocoapods.org</a> , which is the official CocoaPods trunk repository.
Custom URL Base	A read-only field containing the custom base URL defined for the JPD.
External Dependencies Enabled	Defines whether to allow pods with source URLs other than Git to be included in the remote repository.

### 6.3.4.4 | Additional Settings for Composer Remote Repositories

In addition to the basic settings and advanced settings that are common for all remote repositories, the following settings are available when configuring Composer repositories.

Field	Description
Git providers	Select the Git provider to be used by the remote repository: <ul style="list-style-type: none"><li>• GitHub</li><li>• BitBucket</li><li>• Stash/Private BitBucket</li><li>• Stash/Private BitBucket (prior to 5.1.0)</li><li>• Artifactory (used when configuring a Smart Remote repository to another instance of Artifactory)</li><li>• Custom</li></ul>
Registry URL	Defines the base URL of the registry API used by the remote repository. The default is <a href="https://packagist.org">https://packagist.org</a> .

### 6.3.4.5 | Additional Settings for Conan Remote Repositories

In addition to the basic settings and advanced settings that are common for all remote repositories, the following settings are available when configuring Conan repositories.

# JFrog Artifactory Documentation

## Displayed in the header

Field	Description
Force Authentication	Select this option to require the use of basic authentication credentials to use this repository.

### 6.3.4.6 | Additional Settings for Debian Remote Repositories

In addition to the basic settings and advanced settings that are common for all remote repositories, the following settings are available when configuring Debian repositories.

Field	Description
List Remote Artifacts	When set, Artifactory lets you navigate the contents of the repository at the remote registry, even if the artifacts have not been cached in this repository. By default, this option is not set.
Enable indexing with debug symbols (.ddeb)	When set, it enables the indexing of debug symbols for more efficient debugging.

### 6.3.4.7 | Additional Settings for Docker Remote Repositories

In addition to the basic settings and advanced settings that are common for all remote repositories, the following settings are available when configuring Docker repositories.

#### Additional Basic Settings

Field	Description
Project ID	The unique identifier to use for a Docker project and its associated resources. Use this field to enter your GCR/GAR (Google Container Registry/Google Artifact Registry) project ID to limit the scope of this remote repository to a specific project in your third-party registry.  If this field is left blank, remote repositories that support a project ID will default to their default project as set up in your account.
Enable Token Authentication	Enables token-based (bearer) authentication. Select this option when you are proxying the Docker Hub ( <a href="https://registry-1.docker.io/">https://registry-1.docker.io/</a> ).
Block pushing of image manifest v2 schema 1	When set, Artifactory will block the pushing of Docker images with manifest v2 schema 1 to this repository.

#### Additional Advanced Settings

Field	Description
Enable Foreign Layers Caching	When selected, enables Artifactory to download foreign layers to a Docker remote repository. A foreign layer refers to layers within a Docker image that are stored in a different location from the registry from which the image is being pulled.
Patterns Allow List	This optional setting allows you to define include patterns (Ant-style path expressions) to match external URLs when trying to download foreign layers. Supported expressions include (*, **, ?). By default, this field is set to **, which means that foreign layers may be downloaded from any external source.

### 6.3.4.8 | Additional Settings for Generic Remote Repositories

In addition to the basic settings and advanced settings that are common for all remote repositories, the following settings are available when configuring Generic repositories.

#### Additional Basic Settings

Field	Description
List Remote Artifacts	When set, Artifactory lets you navigate the contents of the repository at the remote registry, even if the artifacts have not been cached in this repository. By default, this option is not set.

# JFrog Artifactory Documentation

## Displayed in the header

### Additional Advanced Settings

Field	Description
Propagate Query Params	When set, any query parameters included in the request to Artifactory are passed on to the remote repository.

#### 6.3.4.9 | Additional Settings for Go Remote Repositories

In addition to the basic settings and advanced settings that are common for all remote repositories, the following settings are available when configuring Go repositories.

Field	Description
Git provider	Select the Git provider to be used by the remote repository: <ul style="list-style-type: none"><li>• GitHub</li><li>• GitHub Enterprise</li><li>• Artifactory (used when configuring a Smart Remote repository to another instance of Artifactory)</li><li>• BitBucket Server</li><li>• BitBucket Cloud</li><li>• GitLab</li></ul>

#### 6.3.4.10 | Additional Settings for Helm OCI Remote Repositories

In addition to the basic settings and advanced settings that are common for all remote repositories, the following settings are available when configuring Helm OCI repositories.

### Additional Basic Settings

Field	Description
Enable Token Authentication	When set, token-based (bearer) authentication is enabled.

### Additional Advanced Settings

Field	Description
Enable Foreign Layers Caching	When selected, enables Artifactory to download foreign layers to a Helm OCI remote repository.
Patterns Allow List	This optional setting allows you to define include patterns (Ant-style path expressions) to match external URLs when trying to download foreign layers. Supported expressions include (*, **, ?). By default, this field is set to **, which means that foreign layers may be downloaded from any external source.

#### 6.3.4.11 | Additional Settings for Legacy Helm Remote Repositories

In addition to the basic settings and advanced settings that are common for all remote repositories, the following settings are available when configuring Helm repositories.

### Additional Basic Settings

Field	Description
Charts Base URL	Updates the index.yaml file with the corresponding path where the charts are located on the remote repository. When the Helm client fetches the artifacts, it will fetch them through Artifactory. This field must contain the base path where the charts/tgz files are present.  Use this option when the Helm charts and the index.yaml are stored in different locations. For example, if the remote registry is:

# JFrog Artifactory Documentation

## Displayed in the header

Field	Description
	<p><a href="https://fluxcd-community.github.io/helm-charts">https://fluxcd-community.github.io/helm-charts</a></p> <p>But the charts are stored in:</p> <p><a href="https://github.com/fluxcd-community/helm-charts/releases/download">https://github.com/fluxcd-community/helm-charts/releases/download</a></p> <p>Enter the chart path in this field.</p> <p>For more information about this option, see the Knowledge Base article, <a href="#">ARTIFACTORY: How to configure the "Charts Base URL" in a Helm repository</a>.</p>

### Additional Advanced Settings

Field	Description
Enable Dependency Rewrite	When selected, enables the specification of a "safe" Allow List from which dependencies may be downloaded, cached in Artifactory, and configured to rewrite the dependencies so that the Helm client accesses dependencies through a remote repository.
Patterns Allow List	An Allow 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.  For example, if you limit the Patterns Allow List to <code>https://github.com/**</code> , the external dependencies will be cached in the "helm" remote repository, and only charts with a URL starting with <code>https://github.com/</code> will be allowed to be cached.

#### 6.3.4.12 | Additional Settings for Hex Remote Repositories

In addition to the basic settings that are common for all remote repositories, the following settings are available when configuring Hex repositories.

Field	Description
Public Key	Contains the public key used when downloading packages from the Hex remote registry (public, private, or self-hosted Hex server).
Select Key Pair	Select the RSA key pair to sign and encrypt content for secure communication between Artifactory and the Mix client.

#### 6.3.4.13 | Additional Settings for Maven/Gradle/Ivy/SBT Remote Repositories

In addition to the basic settings that are common for all remote repositories, there are specific settings for the following package types:

- Maven
- Gradle
- Ivy
- SBT

Field	Description
List Remote Artifacts	[Maven only] When set, Artifactory lets you navigate the contents of the repository at the remote registry, even if the artifacts have not been cached in this repository. By default, this option is not set.
Checksum Policy	Checking the Checksum effectively verifies the integrity of a deployed resource. The <b>Checksum Policy</b> determines how the system behaves when a client checksum for a remote resource is missing or conflicts with the locally calculated checksum.  There are four options:

# JFrog Artifactory Documentation

## Displayed in the header

Field	Description
	<ul style="list-style-type: none"> <li>• <b>Generate if absent</b> (default): The system attempts to retrieve the remote checksum. If it is not found, the system will automatically generate one and fetch the artifact.</li> <li>If the remote checksum does not match the locally calculated checksum, the artifact will not be cached and the download will fail.</li> <li>• <b>Fail</b>: If the remote checksum does not match the locally calculated checksum, or is not found, the artifact will not be cached and the download will fail.</li> <li>• <b>Ignore and generate</b>: The system 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.</li> <li>• <b>Ignore and pass-through</b>: The system 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.</li> </ul>
Max Unique Snapshots	<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>
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 Checks	<p>By default, the system keeps your repositories healthy by refusing POMs with incorrect coordinates (path). If the groupId:artifactId:version 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 <b>Suppress POM Consistency</b> 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 the initial access time to the source jar when it is subsequently requested.
Handle Releases	If set, Artifactory enables you to deploy or cache release artifacts into this repository.
Handle Snapshots	If set, Artifactory enables you to deploy or cache snapshot artifacts into this repository.

### 6.3.4.14 | Additional Settings for NuGet Remote Repositories

In addition to the basic settings and advanced settings that are common for all remote repositories, the following settings are available when configuring NuGet repositories.

Field	Description
NuGet Download Context Path	<p>Defines the context path prefix through which NuGet downloads are served.</p> <p>For example, the NuGet Gallery download URL is <a href="https://nuget.org/api/v2/package">https://nuget.org/api/v2/package</a>. Therefore, the repository URL should be configured as '<a href="https://nuget.org">https://nuget.org</a>' and the download context path should be configured as <b>api/v2/package</b>.</p>
NuGet Feed Context Path	Defines the context path prefix for the NuGet feed. The default value is <b>api/v2</b> .
NuGet v3 Feed URL	<p>Defines the URL for the NuGet v3 feed.</p> <p>For example, the feed URL for the official nuget.org repository is (also the default value): <a href="https://api.nuget.org/v3/index.json">https://api.nuget.org/v3/index.json</a></p>
NuGet Symbol Server URL	Defines the URL for the NuGet Symbol Server, which is the central repository for symbol packages used for debugging purposes. The default is <a href="https://symbols.nuget.org/download/symbols">https://symbols.nuget.org/download/symbols</a> .
Force Authentication	Select this option to require the use of basic authentication credentials to use this repository.

### 6.3.4.15 | Additional Settings for Opkg Remote Repositories

In addition to the basic settings and advanced settings that are common for all remote repositories, the following settings are available when configuring Opkg repositories.

## JFrog Artifactory Documentation Displayed in the header

Field	Description
List Remote Artifacts	When set, Artifactory lets you navigate the contents of the repository at the remote registry, even if the artifacts have not been cached in this repository. By default, this option is not set.

### 6.3.4.16 | Additional Settings for P2 Remote Repositories

In addition to the basic settings and advanced settings that are common for all remote repositories, the following settings are available when configuring P2 repositories.

Field	Description
List Remote Folder Items	When set, lists the items of remote folders when browsing (simple and list browsing). This setting is required for dynamic resolution that depends on remote folder content information. The remote content is cached according to the value of the Retrieval Cache Period.

### 6.3.4.17 | Additional Settings for PyPI Remote Repositories

In addition to the basic settings and advanced settings that are common for all remote repositories, the following settings are available when configuring PyPI repositories.

Field	Description
Registry URL	Defines the base URL of the registry API used by the remote repository.
Registry Suffix	Defines the suffix appended to the end of the PyPI base URL. It is usually defined with the default value 'simple' (corresponding to https://pypi.org/simple/). Define a different value if the remote registry is a PyPI server that has a custom registry suffix (for example, +simple in DevPI).

### 6.3.4.18 | Additional Settings for RPM Remote Repositories

In addition to the basic settings and advanced settings that are common for all remote repositories, the following settings are available when configuring RPM repositories.

Field	Description
List Remote Artifacts	When set, Artifactory lets you navigate the contents of the repository at the remote registry, even if the artifacts have not been cached in this repository. By default, this option is not set.

### 6.3.4.19 | Additional Settings for Swift Remote Repositories

In addition to the basic settings and advanced settings that are common for all remote repositories, the following settings are available when configuring Swift repositories.

Field	Description
External Dependencies Enabled	When set, allows external dependencies to be included in the remote repository.
External Dependencies Patterns	This optional setting allows you to define include patterns (Ant-style path expressions) to match those external dependencies to include in the remote repository.

### 6.3.4.20 | Additional Settings for Terraform Remote Repositories

In addition to the basic settings and advanced settings that are common for all remote repositories, the following settings are available when configuring Terraform repositories.

# JFrog Artifactory Documentation

## Displayed in the header

Field	Description
Git providers	Select the Git provider to be used by the remote repository: <ul style="list-style-type: none"><li>• GitHub</li><li>• BitBucket</li><li>• Stash/Private BitBucket</li><li>• Stash/Private BitBucket (prior to 5.1.0)</li><li>• Artifactory (used when configuring a Smart Remote repository to another instance of Artifactory)</li><li>• Custom</li></ul>
Registry URL	Defines the base URL of the registry API used by the remote repository.
Providers URL	Defines the base URL of the provider's storage API used by the remote repository. The default is <a href="https://releases.hashicorp.com">https://releases.hashicorp.com</a> .

### 6.3.4.21 | Additional Settings for VCS Remote Repositories

In addition to the basic settings and advanced settings that are common for all remote repositories, the following settings are available when configuring VCS repositories.

Field	Description
Git providers	Select the Git provider to be used by the remote repository: <ul style="list-style-type: none"><li>• GitHub</li><li>• BitBucket</li><li>• Stash/Private BitBucket</li><li>• Stash/Private BitBucket (prior to 5.1.0)</li><li>• Artifactory (used when configuring a Smart Remote repository to another instance of Artifactory)</li><li>• Custom</li></ul>
Max Unique Snapshots	Defines the maximum number of unique snapshots of a single artifact to store. Once the number of snapshots exceeds this setting, older versions are removed. A value of 0 (default) indicates there is no limit, and unique snapshots are not cleaned up.

### 6.3.5 | Browse Remote Repositories

In some cases, the remote resource for which Artifactory serves as a proxy supports remote browsing. In these cases, you can browse the contents of these repositories directly from the UI.

For example, Maven Central supports remote repository browsing, while Docker Hub does not. In the example below, the contents of Maven Central are displayed.

#### Note

The package displayed above is not cached, meaning that this package exists in Maven Central but is not yet in Artifactory since there have been no requests for this package when resolving dependencies through this repository.

#### 6.3.6 | Handling Offline Scenarios

The system 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, the system can be configured to ignore it by selecting 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

Global Offline Mode 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 **Administration** module under **Artifactory | Settings**.

#### 6.4 | Smart Remote Repositories

# JFrog Artifactory Documentation

## Displayed in the header

A Smart Remote repository is a remote repository that proxies a local, remote, or Federated repository from another instance of Artifactory or an Edge node. 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 can 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 (by 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.

### Download statistics may vary between Artifactory instances

Downloads are reported through the proxy chain only from the time this option is set, so the actual download statistics reported for an artifact may differ between the local Artifactory instance and 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 changed 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 of its artifacts have been cached in your instance of Artifactory. This enables you to browse remote files as if they were local without having to request a particular file.

### Source absence detection

When viewing a cached artifact, Artifactory will indicate if the original artifact in the remote instance has been deleted. This enables you to copy the artifact from your remote repository cache to a local repository to maintain access to it.

### Note

When configuring a Smart Remote Repository, it is recommended that a local or remote Artifactory repository be used as an endpoint. A virtual repository is not recommended, as it might result in inconsistent behavior.

#### 6.4.1 | Configure a Smart Remote Repository

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>`

The repository URL must be prefixed in the path with `api/<type>` for several repository types.

For example,

`http://<JFrog URL>/artifactory/api/<package type>/<repository key>`

When using Artifactory Cloud the URL is:

`https://<server name>.jfrog.io/artifactory/api/<package type>/<repository key>`

The prefix is required for the following repository types:

Type	Prefix
Ansible	<code>api/ansible</code>
Bower	<code>api/bower</code>
Cargo	<code>api/cargo</code>
Chef	<code>api/puppet</code>
CocoaPods	<code>api/pods</code>
Composer	<code>api/composer</code>
Conan	<code>api/conan</code>
Conda	<code>api/conda</code>
Cran	<code>api/cran</code>
Docker	<code>api/docker</code>

Type	Prefix
Gems	api/gems
GitLFS	api/gitlfs
Go	api/go
Helm OCI	api/oci
HuggingFace	api/huggingfaceml
NuGet	api/nuget
Npm	api/npm
OCI	api/oci
Puppet	api/puppet
PyPI	See PyPI Settings
Swift	api/swift
VCS	api/vcs

#### PyPI Settings

PyPi repositories also require a registry URL, which depends on whether the target is a local or remote repository on the target Artifactory instance. For example:

**For a local repository:**

**URL:**`http://<JFROG_URL>/artifactory/pypi-local/`**Registry URL:**`http://<JFROG_URL>/artifactory/api/pypi/pypi-local/`

**For a remote repository:**

**URL:**`http://<JFROG_URL>/artifactory/pypi-remote/`**Registry URL:**`http://<JFROG_URL>/artifactory/api/pypi/pypi-remote/`

After you finish entering the URL and move to another field, Artifactory detects automatically that the remote URL is on another instance of Artifactory and displays a banner above the URL.

The Basic tab of the New Remote Repository screen contains additional fields where you can configure the behavior of your Smart Remote repository.

Field	Description
Report Statistics	If set, Artifactory will notify the remote instance whenever an artifact in the Smart Remote Repository is downloaded locally so that it can update its download counter.  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 local and remote Artifactory instances.
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.  The trigger to synchronize the properties is a download of the artifact from the remote repository cache of the local Artifactory instance.
List Remote Folder Items	If set, enables Remote List Browsing.
Source Absence Detection	If set, Artifactory displays an indication on cached items if they have been deleted from the corresponding repository in the remote Artifactory instance.
Add User Context for Curation	If enabled, Artifactory will include the user's context in requests sent to the configured remote repository. The user's context will be used by Curation for traceability. This context includes the user's email address as well as repository information, such as the repository ID. Please ensure this is compliant with your regulatory and security requirements.

# JFrog Artifactory Documentation

## Displayed in the header

Field	Description
	Relevant for users with JFrog Curation.
Enable Pass-through for Curation Audit	To enable pass-through, first enable <b>Add User Context for Curation</b> . This allows the curation-audit CLI command to download packages from upstream curated remotes. These packages are <i>not</i> cached by the remote repository. Malicious packages are blocked regardless of the pass-through status. For more information, see <a href="#">Configure Curation Pass-Through</a> .

You can modify these settings at any time from the [Edit Repository](#) screen.

### Smart Remote NuGet Repositories

When configuring a Smart Remote NuGet repository, make sure to set **Nuget Download Context Path** to Download and leave the **NuGet Feed Context Path** field blank.

### REST API

For a repository to be identified as a Smart Remote repository, go to the Repository Configuration JSON and under `contentSynchronisation` set the `enabled` flag to true.

#### 6.4.2 | 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. If the remote server discloses its files in a simple HTML index, Artifactory will try to list them.

### 6.5 | Virtual Repositories

To simplify access to different repositories, Artifactory allows you to define a virtual repository. A virtual repository is a collection of local, remote, and other virtual repositories with the same package type 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.

#### 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` file located at `$JFROG_HOME/artifactory/var/etc/artifactory` and set the following flag to `false`:

```
## Disable the download access to the global 'repo'  
artifactory.repo.global.disabled=false
```

This change requires you to restart your Artifactory service.

Once enabled the repository is available at:

`http://<hostname>:<port>/artifactory/repo`

#### 6.5.1 | Configure a Virtual Repository

The procedure for configuring a virtual repository includes tabs for mandatory basic settings and a tab for advanced settings.

##### To configure a virtual repository:

1. In the **Administration** module, select **Repositories**.
2. Click **Create a Repository** and select **Virtual** from the list.

3. In the Select Package Type window, click the icon for the desired package type.

The **Basic** tab for virtual repositories is displayed.

4. In the **Basic** tab, enter the basic settings for the virtual repository. For details, see [Basic Settings for Virtual Repositories](#) and [Select Repositories to Include in a Virtual Repository](#).
5. [optional] In the **Advanced** tab, enter the advanced settings for the virtual repository. For details, see [Additional Virtual Repository Settings for Specific Package Types](#).
6. When you have finished configuring the repository, click **Create Virtual Repository**.

#### Configure a Virtual Repository using the API

Use the **Create Repository** REST API to create a virtual repository. For more information, see [Create Repository](#) and [Repository Configuration JSON](#).

##### 6.5.2 | Basic Settings for Virtual Repositories

The following basic settings are common for all package types.



Setting	Description
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, unique identifier for the repository. It cannot begin with a number or contain spaces or special characters.
Environments	Defines one or more environments in which this repository will reside. Environments aggregate project resources (repositories, Pipeline sources, etc.) to simplify their management. For more information, see Environments.
<b>Note</b>	
Defining an environment is mandatory when creating a repository within a specific project. The default selection is DEV.	
Repository Layout	Sets the layout that the repository should use for storing and identifying modules. A recommended layout that corresponds to the package type defined is suggested.
Public Description	A free text field that describes the content and purpose of the repository. This description can be viewed by all users with access to the repository.
Internal Description	A free text field to add additional notes about the repository. These notes are visible only to the administrator.
Include and Exclude Patterns	The Include Patterns and Exclude Patterns fields provide a way to filter out specific repositories when resolving the location of different artifacts. 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). Example:

# JFrog Artifactory Documentation

## Displayed in the header

Setting	Description
	<p>Consider that the Include Patterns and Exclude Patterns for a repository are as follows:</p> <p><b>Include Patterns:</b> org/apache/**, com/acme/**</p> <p><b>Exclude Patterns:</b></p> <p>com/acme/exp-project/**</p> <p>In this example, the repository is searched for org/apache/maven/parent/1/1.pom and com/acme/project-x/core/1.0/nit-1.0.jar but not for com/acme/exp-project/core/1.1/san-1.1.jar because com/acme/exp-project/** is specified as an Exclude pattern. For more information, see <a href="#">Using Include and Exclude Patterns</a>.</p>
Remote Requests Can Retrieve Remote Artifacts	When set, remote Artifactory instances can retrieve artifacts from remote repositories aggregated under this virtual repository, even if the artifacts are not cached. When not set, artifacts can be retrieved from the caches only (default).  Relevant for all package types <b>except</b> Swift, Terraform, and Composer.
Included Repositories	See <a href="#">Select Repositories to Include in a Virtual Repository</a> .
Default Deployment Repository	Defines the default repository when deploying artifacts to a virtual repository. For more information, see <a href="#">Deploy to a Virtual Repository</a> .

### Important

For information about specific settings for particular package types, see [Additional Virtual Repository Settings for Specific Package Types](#).

### Using Include and Exclude Patterns

The ability to define an **Include Pattern** and an **Exclude 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 and 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. To learn more, see this [Knowledge Base article](#).

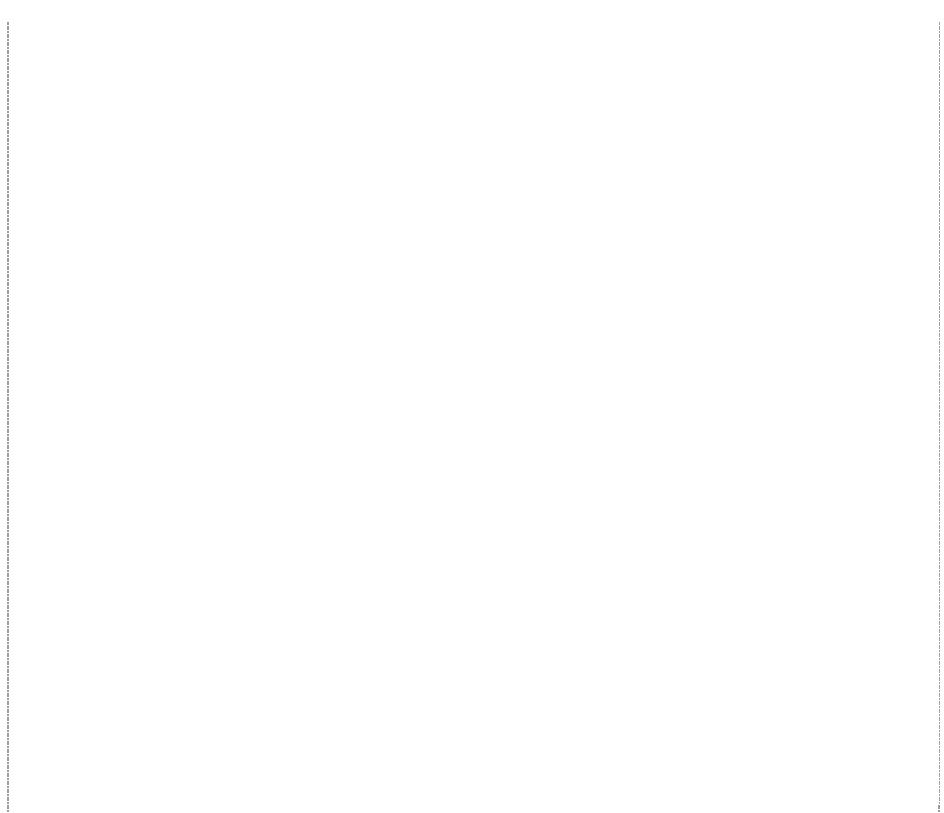
Consider another example in which you wish to define a virtual repository for your developers, however, you wish to keep certain artifacts hidden from them. This could be achieved by defining an **Excludes Pattern** based on groupId, source, or version.

#### 6.5.2.1 | Select Repositories to Include in a Virtual Repository

When creating a virtual repository, you must select the local and remote repositories that will be included. The process is identical for all package types, except for P2 repositories.

##### To select repositories to include in a virtual repository:

1. In the **Administration** module, select **Repositories**.
2. Click **Create a Repository** and select **Virtual** from the list.
3. In the **Select Package Type** window, click the icon for the desired package type. The **Basic** tab for virtual repositories is displayed.
4. Select the Available Repositories you want to include in the new virtual repository and move them to the **Selected Repositories** list.



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.

**Tip**

This list can be re-ordered by dragging and dropping within the Selected Repositories list. This is important as it sets the resolution order.

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 in which they are presented in the **Selected Repositories** list.

When fulfilling a request for the latest version of an artifact from a virtual repository, Artifactory will search all the included repositories to ensure it retrieves the latest version. This means that Artifactory will still search the remote repository even if it finds a version of the artifact in a local or cache repository, to be certain of returning the most current one. If the remote repository is found to have the latest version, Artifactory will download it and update the remote repository cache.

[Virtual Repositories and Projects](#)

Like other resources, a virtual repository can be assigned to a specific project and optionally shared with additional projects. If there are selected local and remote repositories that are not assigned to, or shared with, the same project as the virtual repository, the following message appears in the UI:

Click **View** to see a list of conflicting repositories. Click the export icon to export the list to a CSV file.

Administrators need to be aware of possible repository conflicts to avoid a situation where a user with access to the virtual repository cannot access an aggregated repository because they do not have access to that repository's project.

#### Nesting

Nesting is a unique feature in Artifactory and facilitates more flexibility in using virtual repositories.

### Note

Be careful 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 cannot be resolved due to invalid nesting.

#### 6.5.2.2 | 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](#).

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, see [Configure a Virtual Repository](#).

#### 6.5.3 | Additional Virtual Repository Settings for Specific Package Types

Virtual repositories may have additional settings depending on the package type, as described in the following topics:

- [Additional Settings for Alpine Virtual Repositories](#)
- [Additional Settings for Bower Virtual Repositories](#)
- [Additional Settings for Chef Virtual Repositories](#)
- [Additional Settings for Conan Virtual Repositories](#)
- [Additional Settings for Conda Virtual Repositories](#)
- [Additional Settings for CRAN Virtual Repositories](#)
- [Additional Settings for Debian Virtual Repositories](#)
- [Additional Settings for Docker Virtual Repositories](#)
- [Additional Settings for Go Virtual Repositories](#)
- [Additional Settings for Maven/Gradle/Ivy/SBT Virtual Repositories](#)
- [Additional Settings for npm Virtual Repositories](#)
- [Additional Settings for NuGet Virtual Repositories](#)
- [Additional Settings for OCI Virtual Repositories](#)
- [Additional Settings for P2 Virtual Repositories](#)

##### 6.5.3.1 | Additional Settings for Alpine Virtual Repositories

In addition to the basic settings that are common for all virtual repositories, the following settings are available when configuring Alpine repositories.

Field	Description
Metadata Retrieval Cache Period (sec)	Defines the length of time (in seconds) to cache metadata files before checking for newer versions on aggregated repositories. A value of 0 indicates no caching. The default is 7200 seconds.

##### 6.5.3.2 | Additional Settings for Bower Virtual Repositories

In addition to the basic settings and advanced settings that are common for all virtual repositories, the following settings are available when configuring Bower repositories.

Field	Description
Enable Dependency Rewrite	When selected, external dependencies are rewritten.
Remote Repository for Cache	Defines the remote repository aggregated by this virtual repository in which the external dependency will be cached.

## JFrog Artifactory Documentation Displayed in the header

Field	Description
Patterns Allow List	<p>Defines an Allow List of Ant-style path expressions that specify where external dependencies may be downloaded from.</p> <p>By default, this is set to **, which means that dependencies may be downloaded from any external source.</p> <p>For example, if you wish to limit external dependencies to be downloaded only from <a href="#">github.com</a>, you should add <b>**github.com**</b> (and remove the default ** expression).</p>

### 6.5.3.3 | Additional Settings for Chef Virtual Repositories

In addition to the basic settings that are common for all virtual repositories, the following settings are available when configuring Chef repositories.

Field	Description
Metadata Retrieval Cache Period (sec)	<p>Defines the length of time (in seconds) to cache metadata files before checking for newer versions on aggregated repositories. A value of 0 indicates no caching. The default is 7200 seconds.</p>

### 6.5.3.4 | Additional Settings for Conan Virtual Repositories

In addition to the basic settings that are common for all virtual repositories, the following settings are available when configuring Conan repositories.

Field	Description
Metadata Retrieval Cache Period (sec)	<p>Defines the length of time (in seconds) to cache metadata files before checking for newer versions on aggregated repositories. A value of 0 indicates no caching. The default is 7200 seconds.</p>
Force Authentication	Select this option to require the use of basic authentication credentials to use this repository.

### 6.5.3.5 | Additional Settings for Conda Virtual Repositories

In addition to the basic settings that are common for all virtual repositories, the following settings are available when configuring Conda repositories.

Field	Description
Metadata Retrieval Cache Period (sec)	<p>Defines the length of time (in seconds) to cache metadata files before checking for newer versions on aggregated repositories. A value of 0 indicates no caching. The default is 7200 seconds.</p>

### 6.5.3.6 | Additional Settings for CRAN Virtual Repositories

In addition to the basic settings that are common for all virtual repositories, the following settings are available when configuring CRAN repositories.

Field	Description
Metadata Retrieval Cache Period (sec)	<p>Defines the length of time (in seconds) to cache metadata files before checking for newer versions on aggregated repositories. A value of 0 indicates no caching. The default is 7200 seconds.</p>

### 6.5.3.7 | Additional Settings for Debian Virtual Repositories

# JFrog Artifactory Documentation

## Displayed in the header

In addition to the basic settings that are common for all virtual repositories, the following settings are available when configuring **Debian** repositories.

Field	Description
Metadata Retrieval Cache Period (sec)	<p>Defines how long before Artifactory checks for a newer version of a requested artifact in the repository.</p> <p>A value of 0 means that Artifactory will always check for a newer version.</p> <p><b>On which file types does this parameter work?</b></p> <p>This setting refers to artifacts that expire after a period of time (e.g. metadata files such as maven-metadata.xml, npm package.json or Docker manifest.json etc.).</p> <p>Note that most artifacts that are downloaded do not change (e.g. release versions), therefore this setting does not affect them.</p>
Indexed Remote Architectures	<p>Specifies the architectures that will be indexed for the included remote repositories. Specifying these architectures will speed up Artifactory's initial metadata indexing process.</p> <p>The default architecture values are <b>amd64</b> and <b>i386</b>.</p>
Optional Index Compression Formats	Used for selecting the index file formats to create in addition to the default Gzip (.gzip extension) format, which is created for every Debian repository and cannot be disabled.
Enable indexing with debug symbols (.ddeb)	When set, it enables the indexing of debug symbols for more efficient debugging.
Primary Key Name	The name of the GPG primary key used by the repository.
Secondary Key Name	The name of the GPG secondary key used by the repository

### 6.5.3.8 | Additional Settings for Docker Virtual Repositories

In addition to the [basic settings](#) and [advanced settings](#) that are common for all virtual repositories, the following settings are available when configuring **Docker** repositories.

#### Additional Advanced Settings

Field	Description
Resolve Docker Tags By Latest Timestamp	When enabled, in cases where the same Docker tag exists in two or more of the aggregated repositories, Artifactory will return the tag that has the latest timestamp.

### 6.5.3.9 | Additional Settings for Go Virtual Repositories

In addition to the [basic settings](#) and [advanced settings](#) that are common for all virtual repositories, the following settings are available when configuring **Go** repositories.

Field	Description
Enable 'go-import' Meta Tags	When selected (default), Artifactory will automatically follow remote VCS roots in go-import meta tags to download remote modules.
go-import Allow List	Defines an Allow List of Ant-style path expressions that specify where external dependencies may be downloaded from. By default, this is set to **, which means that dependencies may be downloaded from any external source. For example, if you wish to limit external dependencies to be downloaded only from github.com, you should add <b>**github.com**</b> (and remove the default ** expression).

### 6.5.3.10 | Additional Settings for Maven/Gradle/Ivy/SBT Virtual Repositories

# JFrog Artifactory Documentation

## Displayed in the header

In addition to the basic settings and advanced settings that are common for all virtual repositories, there are specific settings for the following package types:

- Maven
- Gradle
- Ivy
- SBT

Field	Description
Cleanup Repository References in POMs	This setting provides the ability to ensure Artifactory is the sole provider of artifacts in the system by automatically cleaning up the POM (Project Object Model) file. The POM is the XML file that contains configuration and build information about a Maven project.  Available options include: <ul style="list-style-type: none"><li>• <b>Discard active references</b> (default): Removes repository elements that are declared directly under the project or under a profile in the same POM that is <code>activeByDefault</code>.</li><li>• <b>Discard any reference</b>: Removes all repository elements regardless of whether they are included in an active profile or not.</li><li>• <b>Nothing</b>: Does not remove any repository elements declared in the POM.</li></ul>
Key-Pair	Defines a named key-pair to use for signing artifacts automatically.
Force Authentication	Select this option to require the use of basic authentication credentials to use this repository.  This setting is relevant for Maven virtual repositories only.

### 6.5.3.11 | Additional Settings for npm Virtual Repositories

In addition to the basic settings and advanced settings that are common for all virtual repositories, the following settings are available when configuring [npm](#) repositories.

#### Basic Settings

Field	Description
Metadata Retrieval Cache Period (sec)	Defines the length of time (in seconds) to cache metadata files before checking for newer versions on aggregated repositories. A value of 0 indicates no caching. The default is 7200 seconds.

#### Advanced Settings

Field	Description
Enable Dependency Rewrite	When selected, external dependencies are rewritten.
Remote Repository for Cache	Defines the remote repository aggregated by this virtual repository in which the external dependency will be cached.
Patterns Allow List	Defines an Allow 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.  For example, if you wish to limit external dependencies to be downloaded only from <code>github.com</code> , you should add <code>!**github.com**</code> (and remove the default <code>**</code> expression).

### 6.5.3.12 | Additional Settings for NuGet Virtual Repositories

In addition to the basic settings that are common for all virtual repositories, the following settings are available when configuring NuGet repositories.

# JFrog Artifactory Documentation

## Displayed in the header

Field	Description
Force Authentication	Select this option to require the use of basic authentication credentials to use this repository.

### 6.5.3.13 | Additional Settings for OCI Virtual Repositories

In addition to the [basic settings](#) and [advanced settings](#) that are common for all virtual repositories, the following settings are available when configuring OCI repositories.

Field	Description
Metadata Retrieval Cache Period (sec)	Defines the length of time (in seconds) to cache metadata files before checking for newer versions on aggregated repositories. A value of 0 indicates no caching. The default is 7200 seconds.

### 6.5.3.14 | Additional Settings for P2 Virtual Repositories

In addition to the [basic settings](#) that are common for all virtual repositories, the following settings are available when configuring P2 repositories.

Field	Description
Force Authentication	Select this option to require the use of basic authentication credentials to use this repository.
Local Repository	Select the local repository to include in the virtual repository. Typically this will be either a Maven or a Generic repository.
Path Suffix	Defines an optional sub-path inside the local repository where P2 metadata files reside. When left empty, P2 metadata files (content, artifacts, compositeContent, etc.) are assumed to reside directly in the repository root directory. If you have a Tycho repository deployed as a single archive, specify the archive's root path. For example: 'eclipse-repository.zip'.
P2 Repository URL	Defines the URL of the remote repository that contains the P2 metadata files. <b>Note</b> 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).

## 6.6 | Federated Repositories

### Note

This feature is supported with [Enterprise X](#) and [Enterprise+](#) licenses with Artifactory versions 7.18.3 or later.

Federated repositories enable enterprises that operate across multiple sites to maintain a single source of truth for their binaries. A Federated repository functions similarly to a local repository on the local JPD but is grouped logically with other Federated members located on remote JPDs to create a Federation. Artifacts and metadata in one JFrog Platform Deployment (JPD) are mirrored to the Federated repositories in the remote JPDs that together comprise the Federation. Changes made to an artifact on one Federation member are replicated asynchronously to the other members.

Once you have created a Federation, changes made to artifacts on one member of the Federation will be replicated asynchronously to the other Federated members. The Federated repository configuration is also aligned across all members of the Federation.

For example, if **Federated repository A** in "JPD A" and **Federated repository B** in "JPD B" are members of the same Federation, deploying a file to **Federated Repository A** in "JPD A", will trigger a copy of the file action to the **Federated Repository B** in "JPD B".

Similarly, deploying a file to **Federated Repository B** in "JPD B", will trigger a copy of the file action to the **Federated Repository A** in "JPD A". The configuration changes are also synchronized based on the latest update (e.g. latest update to the configuration is propagated to all members of the Federation).

### Important

A file that is replicated from one Federation member (the source) to a different Federation member (the target) cannot be downloaded by users from the target member until it receives the metadata.

# JFrog Artifactory Documentation

## Displayed in the header

The arrival of the metadata can be verified by seeing the file displayed in the tree browser or the database of the target member.

Starting from version 7.74, Cloud customers can confirm the file's arrival by looking for the following log message:

```
"Finished the handle mirror event for repo <repo_key> and message: <message>"
```

### Use Cases

Federated repositories provide Enterprise organizations divided across multiple geographical sites with a single source of truth for their binaries as if they were one seamless unit. They support the following use cases:

- Distributed development teams
- Remote production environment
- Increased artifact availability in a multi-site environment

### Defining Characteristics

Federated repositories have the following characteristics:

- Federated repositories use a persistent queue to ensure that events are maintained even in the event of a system failure or restart.
- Metadata events are updated asynchronously among Federated repositories. However, because of the persistent queue, the repository Federation mechanism ensures that all events will be synchronized among all Federation members. For example, an event might not be sent immediately to a different Federation member due to a server failure or network delays, but the system ensures that the event will be sent eventually.
- There is no dependency among Federation members, nor is there member priority when synchronizing the metadata. This means that a member which is slow to respond or experiencing heavy traffic will not impact the other members.
- When the same file is sent from two locations simultaneously, conflict resolution is performed based on timestamps where the last event received overrides previous events. This is why clock synchronization is critical to the proper functioning of the Federation (see [Setup Prerequisites for Federated Repositories](#)).

### Synchronized Metadata

Federated repositories synchronize the following metadata among members:

Folders	Files
<ul style="list-style-type: none"><li>• length (the size of the folder in bytes)</li><li>• properties</li></ul>	<ul style="list-style-type: none"><li>• checksum</li><li>• properties</li><li>• uploadedBy (see note below)</li><li>• lastFileModified (see note below)</li><li>• deploymentDate (see note below)</li></ul>

### Note

Synchronization of the metadata that was introduced as part of the 7.90.1 release (`uploadedBy`, `lastFileModified`, `deploymentDate`) is controlled by the system property, `artifactory.federated.mirror.events.upload.info.propagate.enabled`. By default, this flag is set to false. Set this flag to true on both the source and target JPDs to synchronize this metadata to other members.

### Settings that are not Synchronized

The following repository configuration settings are **not** synchronized among Federation members:

- proxy
- maxNumberOfThreads
- modificationDate
- blackedOut
- repoLayout
- key
- primaryKeyPairDescriptor
- secondaryKeyPairDescriptor
- xrayConfig
- cdnRedirectConfig
- propertySets
- cdnRedirectRepo
- downloadRedirect

For more information about these settings, see [Repository Configuration JSON](#).

#### 6.6.1 | Federated Topology

Federated repositories employ a topology with two separate but integrated mechanisms, whereby artifact metadata is replicated continuously but separately from the binary content. This is done to enable fast synchronization of the metadata, which is replicated rapidly in bulk, while the binary content is brought asynchronously using a separate mechanism based on workers. Priority can

# JFrog Artifactory Documentation

## Displayed in the header

always be given to specific binaries on user demand as required.

Having the metadata replicated quickly is critical because file properties cannot be uploaded before the file itself is uploaded. This is especially true when working with Docker manifests, which cannot be uploaded before the layers listed in the manifest are uploaded.

Dividing the federation of the metadata and the binaries into two separate mechanisms also provides a solution with excellent scalability.

### 6.6.2 | Artifactory Federation Service

To meet the growing needs of customers, JFrog released the standalone Artifactory Federation Service in Q3 2024 for customers in Cloud environments (support for Self-hosted environments was introduced in January 2025) to ensure the timely synchronization of huge volumes of artifact metadata between customer sites.

#### Customer Benefits

The Artifactory Federation Service offers multiple benefits, including:

# JFrog Artifactory Documentation

## Displayed in the header

- **Scalability:** The service is designed from the ground up to grow as the needs of our customers grow.
- **Resource separation:** Federated repositories employ a topology with two separate but integrated mechanisms, whereby artifact metadata is replicated continuously but separately from the binary content. This enables fast synchronization of the metadata, which is replicated rapidly in bulk, while the binary content is brought asynchronously using a separate mechanism based on workers. Separate mechanisms also enable checksum validation on binaries before they are synchronized, which reduces traffic costs by preventing the copying of files that already exist on the target.
- **Federation monitoring:** A new dashboard enables administrators to quickly understand the health of their organization's repository Federations at a glance. For more information, see [Monitor Federated Repositories using the Dashboard](#).
- **Prioritization:** Users can assign high priority to a limited number of repositories to ensure they receive priority access to system resources. For more information, see [View the Status of All Repository Federations](#).
- **Automatic Federation recovery:** The service features an auto-healing mechanism that can identify synchronization problems between members due to an exhausted queue (a queue that has exceeded the maximum number of attempts to send metadata events to other members), reset the failed events, and retry synchronization. This capability is particularly useful in the event a Full Sync operation is interrupted by a restart of one of the Artifactory instances that host a Federation member.

### 6.6.2.1 | Migrate to the Artifactory Federation Service

Use the JFrog CLI to migrate your repository Federations (including their configuration, states, and events) from the legacy Federation service in Artifactory to the standalone Artifactory Federation Service that was introduced in release 7.104.5. This topic also describes how to return (rollback) to the legacy service, if required.

#### Note

It is recommended to verify that no Full Sync operations are in progress before starting the migration process.

#### Prerequisite

You must activate the Artifactory Federation Service before performing migration. For more information, see [Install the Artifactory Federation Service](#).

#### Install the Plugin

Run the following command to install the migration plugin from the official registry of JFrog CLI plugins:

```
jf plugin install federation-migrator
```

This command installs the plugin and the jar of the migration tool, which performs all required operations with one command.

#### Migration Procedure

Run the following command to execute the migration to the standalone Federation service:

```
jf federation-migrator migrate_rtfs <base url without /artifactory> <access token>
```

#### Note

It can take anywhere from a few seconds to several minutes to complete the operation. You do not need to restart Artifactory after performing migration.

#### Note

For details about generating an access token, see [Access Tokens](#).

#### Verify Migration Status

Perform the following steps to verify that the migration has succeeded:

1. Execute the following API: curl -X GET -uadmin:password "http://{{artifactory\_url}}/artifactory/api/federation/migration/status"

If the service has been enabled, this API will return the status of the migration:

```
MIRATION_STARTED  
MIGRATING_EVENTS_TO_RTFS  
MIRATION_COMPLETED
```

If the service has not been enabled, this API returns: Status not configured

2. Execute the following API to verify that migration has completed: curl -X GET -uadmin:password "http://{{platform\_url}}/federation/rtfs"

The API will return true if migration has completed successfully.

3. The new Federation monitoring dashboard should be available in the platform UI. In the **Administration** module, go to **Monitoring > Federation**. For more information, see [View the Status of All Repository Federations](#).

#### Rollback Procedure

Run the following command to return to the legacy Federation service:

```
jf federation-migrator migrate_rt <base url without /artifactory> <access token>
```

#### Note

It can take anywhere from a few seconds to several minutes to complete the operation. You do not need to restart Artifactory after returning to the legacy Federation service.

#### Tip

For additional information, add --help after each command:

- jf federation-migrator --help
- jf federation-migrator migrate\_rtfs --help
- jf federation-migrator migrate\_rt --help

### 6.6.3 | Setup Prerequisites for Federated Repositories

Setup prerequisites for Federated repositories include:

- Artifactory Versions Must be Identical (prior to 7.49.6)
- JFrog Platform Deployment (JPD) clocks must be synchronized
- Configure a Custom Base URL
- Trust must be established between the JPDs
- Oracle RAC Support

#### Artifactory Versions Must be Identical (prior to 7.49.6)

When using a version of Artifactory older than 7.49.6, you must verify that the same Artifactory version is installed in all the Artifactory instances hosting the members to be included in the Federation.

After all instances have been upgraded to Artifactory release 7.49.6, Artifactory includes multi-version support, which enables the members of a Federation to run different versions of Artifactory, even if the version at one site includes configuration features and values that are not supported on the versions running at other sites. Thanks to multi-version support, future upgrades after 7.49.6 can be performed on one site at a time, eliminating the need for simultaneous upgrades across all locations.

For more information, see [Multi-Version Support](#).

#### JFrog Platform Deployment (JPD) Clocks Must be Synchronized

If the federated repository artifacts are simultaneously updated on two (or more) member federated repositories, the update that is registered last will overwrite the other update(s). Therefore, to ensure consistent, predictable and trackable behavior of the system as a whole, the server clocks of all the machines running Federated members must be synchronized.

#### Configure Custom Base URL

Configure the Custom Base URL in the Administration module, under [General | Settings](#). The Base URL supports detecting member JFrog Deployments (JPDs) in your organization.

#### Applies to Self-Hosted Deployment

The Custom Base URL is relevant to Self-Hosted deployments and not applicable in SaaS.

#### Changing the BaseURL in a Federated Repository Environment

In versions prior to Artifactory 7.55.1, you cannot modify the Base URL if you have Federated repositories set up with remote mirroring. Therefore, you will first need to remove the remote members from the Federation and click [Save](#). This limitation no longer applies in versions 7.55.1 and above. For more information, see [Change the Base URL in Federated Repositories](#).

After changing the base URL, proceed to set up the Federated repositories with the original Federated members. For all the remote members to be populated with new settings, it is recommended to wait for a short period of time between removing the remote members and changing the base URL.

#### Establish Trust Between the JPDs

You will need to establish trust between the JPDs in your Federated repository. The options for enabling trust depend on whether you have a Cloud or a Self-hosted deployment.

##### Cloud Customers

JFrog Enterprise/ Enterprise+ SaaS customers:

- Set up a binding token to create bi-directional trust between the JPDs.

##### Self-Hosted Customers

Choose from one of the following options:

- Set up bindings between relevant JPDs: Using binding tokens enables admins to create trust between managed JFrog Platform Deployments (JPDs) once the JPDs have been added to the Mission Control instance, thus simplifying the setup across JPDs.
- Enable a Circle of Trust: This option is intended for customers who are interested in providing full access to the other JPDs, access which is granted automatically once the Circle of Trust is implemented.

#### Note

Federated repositories that use binding tokens (i.e., no Circle of Trust) require an enabled Mission Control microservice in their Artifactory since the token is created by Mission Control.

#### Oracle RAC Support

##### Note

This section is relevant for customers in Self-Hosted environments running Artifactory release 7.71.16 and later or 7.77.8 or later.

Customers who use Oracle Real Application Clusters (RAC) must configure the following Artifactory system property to support Federated repositories:

`artifactory.oracle.node.events.sequence.is.no.cache`

Setting this property to true enables a converter that fixes the Oracle node events sequence definition for RAC instances.

### 6.6.4 | Federated Repository Best Practices

Setting up Federated repositories on JPDs in a multisite environment impacts your organization's network, servers, and databases. It takes careful consideration and planning to configure a Federation that makes efficient use of available resources and provides timely and accurate synchronization across all Federation members.

This is why JFrog recommends building your repository Federations in stages, with thorough evaluations conducted at each stage to verify your infrastructure is configured properly to handle the demands of running repository Federations.

# JFrog Artifactory Documentation Displayed in the header

JFrog recommends the following best practices when configuring your JPD for repository Federations:

- **Persisted queue age**

This setting determines the amount of time that events remain in the queue before they are purged. The default value is 3 days. This means that if there is a disconnection between two Federation members that lasts longer than 3 days, the older events are purged and can no longer be synchronized. You need to perform a Full Sync operation to re-establish the Federation, which can take a long time to perform.

If you increase this value to more than 3 days, it can greatly increase the size of your database table, which in turn increases costs and affects performance (depending on the number of events generated by each site).

This setting is configured per JPD using the property:

```
artifactory.events.log.cleanup.age.of.entries.to.discard.days
```

It should be set to the same value for each Federation member.

- **Number of metadata workers**

Artifactory uses an internal mechanism (referred to here as a 'worker') to send metadata events in bulk via parallel threads from the source JPD to other Federation members. Each thread is managed by a worker. Each thread can handle the events for one Federated repository at a time. The default value is calculated based on the number of cores in the CPU.

In terms of the Federation, the number of threads required is dependent on the number of repositories that are updated at the same time and the frequency of those updates. If many repositories are updated frequently and at the same time, more threads might be required. A sufficiently powerful server (for example, 64-core) should provide sufficient power to manage 20-40 threads.

To configure the number of metadata workers, set the following property in the `artifactory.system.properties` file (a restart is required):

```
artifactory.persistentQueue.trigger.number.of.threads
```

For more information, see [Artifactory System Properties](#).

- **Number of binary import workers**

Artifactory uses an internal mechanism (referred to here as a 'worker') to import binary events from the source JPD to the target Federation members. The default value is 6.

In terms of the Federation, the number of workers required is dependent on the number of binaries the Federation is expected to handle, their relative size, and the capacity of your network. You need to find the correct balance between the amount of network bandwidth dedicated to importing binaries and the time it will take to import them.

To configure the number of binary import workers, set the following property in `binarystore.xml` (under the Federated provider section):

```
numberOfRemoteImporters
```

## Note

For more information about `binarystore.xml`, see [Binary Provider](#).

- **Number of thread executors**

Thread executors (also known as pools), which are located in the target Federation member, enable the metadata events sent from the source Federation member to be uploaded in parallel while maintaining the correct order of the files.

## Important

The number of thread executors should always be **equal to or greater than** the number of metadata workers, otherwise a bottleneck will result.

To configure the number of thread executors, set the property:

```
artifactory.federated.mirror.events.bulk.max.pools
```

For more information, see [Configure Federated Repositories for Bulk Mirroring and Parallel Processing](#).

## Tip

In addition to increasing the number of metadata workers and thread executors, you may need to increase the following timeout thresholds above their default values (2 min.) if you expect your Federations to carry heavy loads:

- `artifactory.federated.mirror.events.bulk.executor.poll.timeout.minutes`
- `artifactory.federated.mirror.events.bulk.fullSync.executor.wait.timeout.minutes`

Increase these values in proportion to the maximum expected load. For more information about these settings, see [Configure Federated Repositories for Bulk Mirroring and Parallel Processing](#).

- **Number of threads for each executor**

This setting determines how many metadata events for a particular repository can be processed in parallel while preserving the correct order (for example, for Docker events this means uploading all layouts followed by all manifests). This number can be increased by adding threads to each thread executor. The default value is 4.

To configure the number of threads for each executor, set the property:

```
artifactory.federated.mirror.events.bulk.threadsPerPool
```

For more information, see [Configure Federated Repositories for Bulk Mirroring and Parallel Processing](#).

- **Network infrastructure**

The capabilities of your organization's network are of paramount importance to the optimal functioning of your Federation. The stronger the network connections between the members and the greater the bandwidth, the better the Federation will be able to avoid latency when sending metadata events and binaries from site to site.

## 6.6.5 | Set Up a Federated Repository

A Federated repository can be set up using:

- Platform UI
- REST APIs

**Important**

Verify that you have met all setup prerequisites before proceeding. Self-hosted customers must configure a Custom Base URL. For more information, see [Setup Prerequisites for Federated Repositories](#).

**Important**

Only Platform Administrators can set up a Federated repository and add members to the Federation.

6.6.5.1 | [Set Up a Federated Repository Using the UI](#)

Setting up a Federated repository is similar to setting up a local repository with the additional step of choosing the repositories on other JPDs that are part of the same Federation.

**Important**

Verify that you have met all setup prerequisites before proceeding. Self-hosted customers must configure a Custom Base URL. For more information, see [Setup Prerequisites for Federated Repositories](#).

To set up a Federated repository using the platform UI:

1. From the **Administration** module, select **Repositories**.
2. Click **Create a Repository** and select **Federated**.

- 
3. Select the package type.
  4. In the **Basic** tab, configure the basic repository settings. For more information, see [Basic Settings for Local Repositories](#).

**Important**

It is mandatory to assign a Repository Key that will be added as the prefix to the Federated repository and displayed on all the sites.

5. In the **Advanced** tab, configure additional repository settings as required, including defining a proxy (if your organization requires you to go through a proxy to access a Federated repository). For more information, see [Basic Settings for Remote Repositories](#).

6. In the **Federation** tab, add members to the Federation by selecting repositories on other JFrog Platform Deployments (JPDs). Click **Add Repository**.

7. In the Add Repositories dialog box, add members to the Federation using one of these methods:

- **Deployments:** If you have JFrog Mission Control installed, the repositories on the remote JPDs will automatically be populated. If the repository with the identical name doesn't exist on the remote JPD, click **Create New** to duplicate this repository.
- **URL:** Manually add a predefined URL path to the repository. If the repository does not exist on the remote JPD, it will be created automatically according to the following syntax:  
`<BASE_URL>/artifactory/<repository_name>` //For example, `http://<ip address>:8082/artifactory/fed117`

**Note**

JPDs will appear only if you have set up bindings or established a Circle of Trust.

8. Click **Done**. You will be returned to the Federated tab.

9. Click **Create Federated Repository**.

#### Federated Repositories Not Supported for Artifactory Backend Repositories

It is not possible to connect a Terraform Backend repository to a Federated repository. This limitation prevents inconsistencies in the system state, which could lead to unexpected behavior or errors.

##### 6.6.5.2 | Set Up a Federated Repository Using the REST API

**Important**

Verify that you have met all setup prerequisites before proceeding. Self-hosted customers must configure a Custom Base URL. For more information, see [Setup Prerequisites for Federated Repositories](#).

The dedicated Federated Repository Configuration JSON file is `application/vnd.org.jfrog.artifactory.repositories.FederatedRepositoryConfiguration+json`

The following repository REST APIs support working with Federated repositories:

- Get Repositories
- Create Repository
- Update Repository Configuration
- Delete Repository

**Note**

When working inside a specific project, the Federated repositories that comprise the Federation must all share the same project key.

##### 6.6.5.3 | Convert a Local Repository to a Federated Repository

Local repositories can be converted to Federated repositories using one of the following methods:

- Platform UI
- REST API

##### 6.6.5.3.1 | Convert a Local Repository to a Federated Repository - Platform UI

You can convert an existing local repository to a Federated repository directly from its listing in the Local tab.

**Warning**

Repository federations and replication cannot co-exist in the same repository. Therefore, if the local repository has been set up previously for repository replication, these replication definitions are deleted automatically when the local repository is converted to a Federated repository.

**To convert a local repository to a Federated repository:**

1. In the Administration module, select **Repositories > Local**.
2. At the end of the row for the relevant local repository, open the Actions menu and select **Convert to Federated**.

The Convert to Federated Repository dialog opens.



3. Click **Convert**.

4. [optional] Add other members to the Federated repository, as described in Set Up a Federated Repository Using the UI.

#### Converted Local Repositories Remain Federated

Removing the repository from the Federation does not automatically revert the repository to a local repository. Removing a repository from the Federation simply disconnects the bi-directional sync; however, the repository remains Federated.

The Federated repository cannot be converted back to a local repository.

##### 6.6.5.2 | Convert a Local Repository to a Federated Repository - REST API

You can convert a local repository to a Federated repository using the Convert Local Repository to a Federated Repository REST API.

#### Converted Local Repositories Remain Federated

Removing the repository from the Federation does not automatically revert the repository to a local repository. Removing a repository from the Federation simply disconnects the bi-directional sync; however, the repository remains Federated.

Use the Convert Federated Repository to a Local Repository REST API to convert back to local, if required.

##### 6.6.5.4 | Convert a Federated Repository to a Local Repository

You can convert a Federated repository back to a local repository using the Convert a Federated Repository to a Local Repository REST API.

#### Important

The conversion from Federated back to local is allowed only if the Federated repository is not part of an active Federation containing additional members.

##### 6.6.5.5 | Convert a Build-Info Repository to a Federated Repository

Build-Info repositories can be set as a federated member in a Federation.

From Artifactory 7.35, you can convert an existing Build-Info repository to a Federated repository using the Artifactory REST API.

- Step 1: Convert the build-info repository to a Federated repository
- Step 2: Add members to the build-info Federation

#### Note

The Federated repository cannot be converted back to a local Build-Info repository.

# JFrog Artifactory Documentation Displayed in the header

## Step 1: Convert the Build-Info Repository to a Federated Repository

```
POST /api/federation/migrate/{buildInfoRepoName}
```

### Sample Input

```
POST /api/federation/migrate/artifactory-build-info
Migration finished successfully.
```

## Step 2: Add Members to the Build-Info Federation

After you have converted the Build-Info repository into a Federated Build-Info repository, add members to the Federation using the Update Repository Configuration REST API.

```
{
  "key": "projectKey-build-info",
  "rclass" : "federated",
  "members": [
    { "url": "http://<target-JPD>/artifactory/projectKey-build-info", "enabled":"true"},
    { "url": "http://<target-JPD>/artifactory/projectKey-build-info", "enabled":"true"}
  ]
}
```

For more information, see [Repository Configuration JSON](#).

### 6.6.6 | Working with Federated Repositories

A Federation is a collection of Federated repositories in up to 10 JFrog Platform Deployments (JPDs) at different sites that are automatically configured for full bi-directional replication. Once you have set up the Federation, changes made to artifacts on one site will be automatically synchronized to the other JPDs in the Federation using bi-directional mirroring.

You can perform the following Federated repository procedures:

- [Working with Federated Artifacts](#)
- [Pause/Resume Federated Synchronization](#)
- [Remove Repositories from a Federation](#)
- [Federation Recovery and Auto Healing](#)
- [Perform Full Sync on Federated Repositories](#)
- [Use the Federation Comparison Tool](#)
- [Configure Federated Repositories for Bulk Event Mirroring and Parallel Processing](#)
- [Configure In-Memory Sorting for Full Sync Operations](#)
- [Geo Synchronized Topology Use Case: Setting a Federated Base URL](#)
- [Change the Base URL in Federated Repositories](#)
- [Increase the Predefined Socket Timeout for Larger Repositories](#)
- [Troubleshoot Federated Member Out-of-Sync Notifications](#)

#### 6.6.6.1 | Working with Federated Artifacts

Once you have created the Federation, the Federated repositories are automatically displayed in your artifact browser.

Any of the CRUD actions that you perform are automatically applied to the member Federated repositories. Users can perform actions according to their permission sets. The logic is applied according to the last action performed on the artifact by any of the users in the Federation.

The importance of knowing which action occurred last is the reason why clock synchronization is critical to the proper functioning of the Federation (see [Setup Prerequisites for Federated Repositories](#)).

#### 6.6.6.2 | Pause/Resume Federated Synchronization

You can pause and resume synchronization of the artifacts from your Federated repository to the other member repositories.

When you pause a Federated repository, the other members of the Federation will not synchronize artifacts and metadata with the disabled member until you resume synchronization.

#### 6.6.6.3 | Remove Members from the Federation

# JFrog Artifactory Documentation

## Displayed in the header

Admins of any Federated members can remove themselves and other members from the Federation. Note that you can remove the initial repository and the Federation will continue to function between the remaining Federated members.

1. From the **Administration** module, click **Repositories > Repositories** and click the **Federated** tab to view a list of Federations.
2. Select the Federation repository from the list and click the **Federation** tab.
3. Click the **x** located on the top right of the repository to be removed.

4. Click **Save**.

### Federation Member Removal – Legacy Service vs. Artifactory Federation Service

There is an important difference in the behavior of the Artifactory Federation Service and the legacy Federation service when you try to remove a member when a different member of the same Federation is offline (for example, due to network connectivity issues or if the remote JPD is down):

- **Legacy service:** If member A tries to delete member B, the system allows the removal even though member C is offline. However, when member C comes back online, it will not be aware of the removal of member B. This requires you to make manual changes on the target JPD to bring member C back in sync with the rest of the Federation.
- **Artifactory Federation Service (RTFS):** You cannot remove a member from the Federation if any other members are offline. Instead, you must first remove the offline members, as shown above.

#### 6.6.6.4 | Federation Recovery and Auto-Healing

##### Important

This feature requires **all** JPDs in the Federation to run Artifactory release 7.71.1 or later.

In certain cases, it may not be possible to maintain near real-time synchronization of all artifact events (create, update, delete) and binary tasks among Federation members. Examples include short-term networking issues between the JPDs, Artifactory upgrades, a user-initiated synchronization pause, and so on. If synchronization continues to fail after reaching the maximum number of retry events, event sync is paused and the Federation moves into an error state.

One way to recover the Federation is to perform a full sync, but this can be a time-consuming process if the Federated repositories contain a large number of artifacts, as this amounts to restarting the Federation.

Artifactory features an auto-healing mechanism that ensures the reliability and consistency of Federated repositories by performing recovery actions at regular intervals. These recovery actions include:

- **Recovering exhausted queues:** The auto-healing mechanism checks for queues that have exceeded the maximum number of attempts to send events to other Federation members. Such queues are considered exhausted. The auto-healing mechanism resets the failed events automatically, enabling the system to retry synchronization with the target mirror.
- **Recovering exhausted binary tasks:** In cases where binary tasks have reached their maximum retry attempts (default: 10 retries), the auto-healing mechanism invokes the Replay Failed Binary Tasks API. This API resets the retry count for the task, enabling the system to retry the task and ensure that binary synchronization tasks are completed successfully.

##### Note

If events have accumulated over a period of days, the event cleanup mechanism might potentially clean events that have not been propagated, causing the queue to move to an out-of-sync state. In such cases, performing a full sync is required.

##### Email Notifications

All administrators who are registered for Artifactory's mail service will receive notifications similar to the one shown below when auto-healing takes place:

```
2023-09-21T10:39:24.696Z [jfret] [INFO] [29cb8b34b3ec63e4] [atedRepositoryRecoveryTest:247] [TestNG_1] ] [rt_229036255] [rt_229036255] - Mail notification subject: [JFrog] Mirror Recovery in Progress  
2023-09-21T10:39:24.696Z [jfret] [INFO] [29cb8b34b3ec63e4] [atedRepositoryRecoveryTest:249] [TestNG_1] ] [rt_229036255] [rt_229036255] - Mail notification content: -  
-----=_Part_0_2088885210.1695292764495  
Content-Type: text/html; charset=us-ascii  
Content-Transfer-Encoding: 7bit  
  
"Auto Healing" task has recognized a mirror in an exhausted state: 'http://localhost:11552/artifactory/generic-fed-9ac0b669-5760-4ff3-a209-c6b8c7826928' ->  
'http://localhost:55295/artifactory/generic-fed-9ac0b669-5760-4ff3-a209-c6b8c7826928'.  
Recovery attempt is now in progress...
```

##### System Properties

Federation recovery and auto-healing are controlled using the following properties in the `artifactory.system.properties` file:

Property	Description
artifactory.auto.healing.job.interval.sec	Defines the interval (in seconds) at which the auto-healing feature checks for exhausted queues.  The default value is 20 seconds.
artifactory.federated.subsequent.event.grace.period	Defines the buffer that works in conjunction with the federated. negotiation.enabled setting.  The default value is 60 seconds.
artifactory.federated.event.queue.max.error.retries	Defines the number of attempts to send a queued Federated event before the queue becomes exhausted and therefore eligible for auto-healing.  The default value is 6.
artifactory.persistentQueue.max.lock.lease.time.minutes	Defines the delay interval (in minutes) between attempts to trigger the queue.  The default value is 1.
artifactory.reset.stale.full.sync.job.interval.min	Defines the interval (in minutes) for an async task that resets the status of a Full Sync operation that has become "stuck", enabling the Full Sync to restart.  This property is useful, for example, if the Artifactory instance is restarted while a Full Sync operation is running. After the restart, this async task will reset the operation and restart it.  The default value is 15.
artifactory.reset.stale.full.sync.job.initial.delay.min	Defines the initial delay (in minutes) before running the async task that resets the status of a stuck Full Sync operation.  The default value is 1.

#### Manual Recovery using a REST API

Use the Federation Recovery REST API to perform recovery manually. This API can be used when auto-healing has been disabled or when you want to perform recovery immediately without waiting for the auto-healing interval to arrive.

##### 6.6.6.5 | Perform Full Sync on Federated Repositories

###### Tip

Users who are running Artifactory 7.7.1 or later on **all** their JPDs should use the auto-healing mechanism (or perform Federation recovery manually using the REST API) before resorting to performing a Full Sync.

In certain cases, it might not be possible to synchronize all the metadata events between repository Federation members. For example, suppose two members are disconnected for a long period of time (for example, more than 3 days). In that case, old events might be deleted from the queue making it impossible to copy those events to the remote member after restoring the connection.

When regular synchronization is incomplete, the solution is to perform a unidirectional Full Sync operation between the repositories. The Full Sync operation synchronizes all the events in one JPD by copying any missing files to designated remote JPDs. However, Full Sync does not synchronize any deleted events.

For example, suppose a file was deleted from JPD A, but the event wasn't propagated to the remote site (JPD B) before the event was deleted from the queue. If you run Full Sync on JPD B (where the file deleted from JPD A still exists), the operation will restore the file on JPD A.

Therefore, if you need to synchronize the entire Federation, you must run the Full Sync operation on all members.

Bear in mind that a Full Sync operation can be time-consuming if the Federated repositories contain many artifacts, as this amounts to restarting the Federation.

A Full Sync operation is performed using the Federated Repository Full Sync REST API. You can use this API to synchronize specific Federated repositories or all the Federation repositories in the Federation.

###### Note

For more information about configuring Full Sync operations, see:

- [Configure Federated Repositories for Bulk Mirroring and Parallel Processing](#)
- [Configure In-Memory Sorting for Full Sync Operations](#)

###### How Full Sync Works

When you initiate a Full Sync operation, the query is sent to the remote member, which begins compiling a file list of the repository's contents. The local JPD polls the remote JPD every 30 seconds (this interval is configurable - see the note below) to check whether the file list is ready. The advantage of using this polling method is that it removes the need to keep the connection between the members open for a long period of time, which in the case of large repositories can lead to timeouts.

When ready, the remote JPD sends its file list to the local JPD for comparison with the local file list, and then the Full Sync operation can begin.

# JFrog Artifactory Documentation

## Displayed in the header

### Note

The following property defines the polling interval:

```
federatedFullSyncPoolingWaitTime("federated.full.sync.pooling.wait.time.sec", TimeUnit.SECONDS.toMillis(30)),
```

### Federated Comparison Tool

The REST API contains an option to use the Federated Comparison tool, which performs a dry run of a Full Sync operation and returns the results in a JSON file. For more information, see [Use the Federation Comparison Tool](#).

### Note

You must have Artifactory 7.104.2 or later and the Artifactory Federation Service installed to use the Federation Comparison tool.

6.6.6.1 | [System Properties for Full Sync File List Queries](#)

The following Artifactory system properties define how to execute file list queries for Full Sync operations on [Federated repositories](#).

The property described in the table below defines how to perform file list queries.

Property	Description
artifactory.federated.full.sync.use.aql	When set to true (default), Artifactory uses a single AQL query to create the Full Sync file list.
	When set to false, Artifactory uses one or more SQL queries to create the Full Sync file list.

The properties in the table below are relevant only when `artifactory.federated.full.sync.use.aql` is set to false.

Property	Description
artifactory.federated.full.sync.paging.threshold	If the number of artifacts for the file list is greater than this threshold, the file list is generated from multiple queries (paging).  If the number of artifacts is smaller than this threshold, the data for all artifacts is loaded using a single query.  The default value is 400000.
artifactory.federated.full.sync.page.size	When using paging to generate the Full Sync file list, this property defines the page size, which means the number of rows read by each database query.  The default value is 100000 rows.
artifactory.federated.full.sync.pages.per.file	Defines the number of pages placed in a single, temporary file to allow efficient sorting of the data.  The default value is 4.

6.6.6.6 | [Use the Federation Comparison Tool](#)

### Note

This feature requires Artifactory 7.104.2 or later and the Artifactory Federation Service must be installed.

The Federation Comparison tool is a validation mechanism for Federated repositories. Instead of actively synchronizing artifacts, this tool compares the state of a Federated repository with one or more remote members to detect missing artifacts in those remote members.

The Federation Comparison tool enables you to verify that all necessary artifacts are properly synced between Federation members, helping to identify any discrepancies due to event loss or system issues.

To use the tool, set the `isDry` query parameter in the Full Sync REST API to true, as shown below:

```
curl -X POST "https://<your-artifactory-instance>/api/federation/fullSync/testDryFullSyncRepo?mirror=https://<mirror-instance>/artifactory/my-repo&isDry=true" \
-H "Authorization: Bearer <your-token>" \
-H "Content-Type: application/json"
```

The JSON report generated by the operation lists all the files that are present in the local member but missing from the remote members. The report is stored in the **jfrog-full-sync-info** system repository. The API response provides the path to the report.

### Sample response:

```
{
  "message": "Dry full sync completed successfully.",
  "resultPaths": [
    "jfrog-full-sync-info/comparison-results/testDryFullSyncRepo-mirror-comparison-2025-02-04.json"
  ]
}
```

# JFrog Artifactory Documentation

## Displayed in the header

Sample report contents:

```
{  
  "results": [  
    {  
      "name": "testDryFullSyncFile_1.txt",  
      "type": "file",  
      "fullPath": "folder/testDryFullSyncFile_1.txt"  
    },  
    {  
      "name": "testDryFullSyncFile_2.txt",  
      "type": "file",  
      "fullPath": "folder/testDryFullSyncFile_2.txt"  
    }  
  ]  
}
```

For complete details about the REST API, see [Federated Repository Full Sync](#).

### 6.6.6.7 | Configure Federated Repositories for Bulk Mirroring and Parallel Processing

Bulk event mirroring, introduced in Artifactory version 7.63.2, helps to optimize Federation performance.

Before bulk event mirroring was introduced, artifacts were copied to target mirrors sequentially one event at a time per repository. This limited the number of events that could be processed in a reasonable amount of time and sometimes resulted in HTTP network bottlenecks, especially between geographically distant members where network latency can be a significant factor.

Bulk event mirroring bypasses potential HTTP network bottlenecks by collecting events into bulks, each of which is dedicated to a single repository, and sending them to the target mirror.

The target mirror assigns a dedicated thread pool to each bulk. Each pool contains multiple threads that upload artifacts to the local Artifactory in parallel. This parallel processing enables events to be uploaded much faster than was previously possible.



In addition to improving the standard event flow between Federation members, as described above, bulk events also improve the performance of Full Sync operations. Working in conjunction with a special cache mechanism (also introduced in 7.63.2), bulk mirroring helps to greatly reduce the load on Artifactory during Full Sync operations.

The following Event Queue properties can be configured for bulk event mirroring:

Property	Description
<code>artifactory.federated.mirror.events.bulk.exclude.repo.types=</code>	<p>Excludes specific package types from bulk event mirroring.</p> <p>To exclude all package types:</p> <p><code>federated.mirror.events.bulk.exclude.repo.types=*</code></p> <p>To exclude specific package types, separate the types with a comma:</p> <p><code>=docker,composer,npm</code></p>
<code>artifactory.federated.mirror.events.bulk.max.pools</code>	<p>The number of available event processing thread pools (executors).</p> <p>Default: 4</p>

# JFrog Artifactory Documentation

## Displayed in the header

Property	Description
artifactory.federated.mirror.events.bulk.threadsPerPool	Maximum: 5 The number of threads per pool (executor). Default: 4
artifactory.federated.mirror.events.batch.message.bulk.size	Maximum: 5 The number of Event Queue events sent with each replication message to the target mirror. Default: 100
artifactory.federated.mirror.events.bulk.executor.poll.timeout.minutes	The amount of time new requests will wait (in minutes) if all processing pools are occupied. Default: 2

Bulk fetching of event properties from the database was introduced in Artifactory release 7.100.x as an additional performance enhancement. The following property can be configured:

Property	Description
artifactory.federated.bulk.properties.enabled	Determines whether bulk property fetching is enabled. Default: false (disabled)

The following Full Sync event properties can be configured for bulk event mirroring:

Property	Description
artifactory.federated.mirror.events.bulk.fullSync.threadsPerPool	The number of threads per pool. Full Sync events take priority in a single dedicated executor containing the number of threads defined here. Default: 10 Maximum: Determined by the number of available processors.
artifactory.federated.mirror.events.bulk.fullSync.executor.wait.timeout.minutes	The amount of time (in minutes) that Full Sync calls for other repositories wait while the pool works on the Full Sync of a particular repository. Default: 2
artifactory.full.sync.batch.retry.count	The number of times to retry a failing batch (bulk) when performing Full Sync. Default: 5
artifactory.full.sync.batch.retry.quiet.period.seconds	The amount of time to wait between batch (bulk) retries. Default: 5 seconds
<b>Note</b>	
This event property requires Artifactory 7.77.3 or above.	

### 6.6.6.8 | Configure In-Memory Sorting for Full Sync Operations

Full Sync queries can take a long time to perform, and under extreme circumstances can sometimes fail to complete. Users who determine that their database is not sufficiently robust or scalable to accommodate heavy loads have the option to sort queries in memory instead of in the database. The sort is performed according to depth, path, and name.

Use the following system properties to configure in-memory sorting:

Property	Description
artifactory.federated.repo.full.sync.enabled.inMemorySorter	Enables in-memory sorting. Default value is false (sorting performed in the database).

# JFrog Artifactory Documentation

## Displayed in the header

Property	Description
artifactory.federated.repo.full.sync.limit.entries.for.split	Determines how many artifacts in the file list are sorted at one time. (This property is relevant only when in-memory sorting is enabled.) Default value is 400000.

### 6.6.6.9 | Geo Synchronized Topology Use Case: Setting a Federated Base URL

The geo-synchronized topology is an extension of the full mesh topology whereby several Artifactory instances are connected to a GeoDNS. In this use case, the desired outcome is to have the exact same configuration (repository names, users, groups, permission targets) in all of the instances connected to the routing server. Users can then deploy and resolve from the same repositories without the need to change the configuration in their build tool according to the server to which they are routing (for example, to divide a load among multiple instances in different locations). For these users, everything is behind the scenes and they just connect to Artifactory through one dedicated URL.

#### Note

Federated repositories do not operate synchronously, and therefore it is recommended to work with one fixed site. If the decision is made to move between one site and another, you must take into account that it takes time for all events to be synchronized between those sites.

#### Setting an Initial Federated Repository URL

A geo-synchronized topology requires that all platform deployments (Artifactory instances) be configured with the same Custom Base URL. However, Federated repositories also need a unique Federated Base URL to distinguish between the Federated members in the different platform deployments.

To work with Federated repositories within a geo-synchronized topology, add the `federatedRepoUrlBase` parameter to the Global Configuration Descriptor file, which is the global Artifactory configuration file used to provide a default set of configuration parameters.

The following XML tag should be added under the `<systemProperties>` section in the Global Configuration Descriptor:

```
<urlBase>https://<ART_LB_URL>/urlBase<!-- NOTE: Enter your Geo-LB URL -->
<federatedRepoUrlBase>https://<ARTIFACTORY_SERVER_HOSTNAME>/artifactory</federatedRepoUrlBase>
```

For information on how to modify the Global Configuration Descriptor, see the Configuration Files Overview.

#### Important

The `federatedRepoUrlBase` described above is relevant **ONLY** for Federated repositories and cannot be applied to other features. For a detailed guide that describes how to set the `urlBase` and still allow redirects and metadata to map to the Geo-LB, see [Geo-Location LB Setup Steps](#).

#### Migrating Federated Repositories to a Geo-Synchronized Topology

From Artifactory version 7.21.13, you can migrate your existing Federated repositories to be configured with a dedicated Federated Repository URL instead of the Artifactory base URL.

#### Migrating in an HA Environment

When setting up Federated Repositories in an HA environment, ensure on both sites that only one HA node is up and running

#### To migrate Federated repositories to a geo-synchronized topology:

1. Disconnect all the Federated repositories in either one of the sites, using one of the following methods:
  - One at a time on the Federated Repository page in the UI
  - Directly through the Config Descriptor file.
2. Wait for a while and then validate that all the Fed repositories are disconnected on the member site.
3. Navigate to `$JFROG_HOME/artifactory/var/etc/artifactory/artifactory.config.xml` and add the following XML tag.

```
<federatedRepoUrlBase>https://<address>/artifactory</federatedRepoUrlBase>
```
4. Reconnect all the Federated repositories.
5. Validate that the configuration was applied in the member site.
6. Monitor the logs for related errors.
7. [Test] Upload a new file for each Repo to site 1 and download it to site 2.
8. [Optional] Perform full synchronization if during the time that the Fed Repos were disconnected changes occurred on one of the sites (For example upload, override, change in properties, or delete actions).

### 6.6.6.10 | Change the Base URL in Federated Repositories

Administrators can change the base URL in an active Federated repository with remote members. Changing the base URL might be necessary, for example, if the network topology has changed.

#### Important

This feature is available only when **all** Federation members are using version 7.55.1 or above.

The steps required to complete the change are dependent on whether the Federation uses the base URL or a special Federated base URL.

If a Federated base URL has been defined, the JPD publishes this address to all Federation members, who then use this address to access the Federated repository on the JPD instead of the standard base URL.



To determine whether a Federated base URL has been defined, continue with [Locate the Federated Base URL](#).

### Important

As a fallback, it is possible to change the base URL more aggressively by canceling the Federation, changing the base URL, and then recreating the Federation. Please note this has a huge impact on the system since it requires recreating each Federated repository from scratch.

### Possible Use Cases

When changing the base URL in Federated repositories, there are two possible use cases:

- If you want to change the base URL, perform [Change the Base URL](#).
- If you want to change the existing **Federated** base URL, perform [Change the Federated Base URL](#).

In both cases, it is highly recommended to test the functionality of the Federation after changing either the base URL or the Federated base URL, as described in [After Changing the Base URL](#).

### Change the Base URL

Administrators can change the base URL directly from the JFrog platform UI, as described below.

1. In the Administration module, select **General > Settings**.
2. Enter a new custom base URL.
3. Click **Save**.

4. Continue with [After Changing the Base URL](#).

**Note**

You can also change the custom base URL using the [Update Custom URL Base REST API](#).

6.6.6.10.1 | [Change the Federated Base URL](#)

If the Federation uses a Federated base URL, the Administrator can make changes to it directly using the Config Descriptor.

**To change the Federated Base URL:**

1. In the Administration module, select **Artifactory > Advanced > Config Descriptor**.

2. Search the configuration file for a Federation base URL entry:



3. Make the required changes to the Federation base URL and save your changes.

6.6.6.10.2 | [Generate New Tokens for Federation Members](#)

When the Federation members do not have a Federated base URL, or if the Federated base URL has changed, the Administrator must generate a new pairing token and pair the source and target JPDs.

**Important**

Make sure to perform this procedure on each remote Federation member.

**To generate new tokens:**

1. Generate a new token from the source JPD using the Create Pairing Token Platform REST API.

# JFrog Artifactory Documentation Displayed in the header

2. Pair the target JPD to the source JPD using the new token with the Initiate Pairing Platform REST API.

3. Replace the base URL (or Federated base URL) by executing the following POST request against the target JPD: `api/federation/replaceUrl` with the following body:

```
{  
    "oldBaseUrl": "http://<old base URL>/artifactory",  
    "newBaseUrl": "http://<new base URL>/artifactory"  
}
```

4. Continue with [After Changing the Base URL](#).

## 6.6.6.10.3 | After Changing the Base URL

It is strongly recommended to test the functionality of the Federation after changing the base URL or Federated base URL. After verifying that the Federation members are synchronized, the administrator can optionally revoke the obsolete keys.

### To test the functionality of the Federation after changing the base URL:

1. In the Administration module, select **Monitoring > Federation Status**. If you see that the Federation is no longer in sync, there is a problem with the new configuration. See [View Federation Sync Status](#) for more information.
2. [optional] After verifying that all Federation members have synchronized successfully with the updated base URL, revoke the obsolete key for each remote JPD using the [Revoke Token by ID REST API](#).

## 6.6.6.11 | Increase the Predefined Socket Timeout for Larger Repositories

From Artifactory version 7.38.17 (Self-Hosted), you can increase the socket timeout when syncing large repositories between federated members within a Federation.

To increase the timeout time, increase the default value 20000 milliseconds using the `artifactory.mirror.http.client.socket.timeout.milli=20000` parameter in the `artifactory.system.properties` file.

## 6.6.6.12 | Troubleshoot Federated Member Out-of-Sync Notifications

If any members of your Federation are out of sync, the system triggers an out-of-sync notification. This can occur due to network failures or changes to your Federated members.

You can manually push the updated configuration to the target Federated member, using one of these methods:

- Run the Synchronize Federated Member Configuration REST API.  
`POST http://localhost:port/artifactory/api/federation/configSync/<repositoryKey>`
- Directly in the JFrog Platform UI:
  1. In the **Administration** module, go to **Repositories > Federated**.
  2. Select the relevant Federated member, and from the Actions menu, select **Push Configuration**.



## 6.6.7 | Monitor Federated Repositories

You can monitor the status of Federated repositories using a variety of tools, including:

- Dedicated REST APIs
- Open Metrics
- Platform UI tools

The available monitoring tools are dependent on whether you are using the standalone [Artifactory Federation Service](#) (introduced for Cloud environments in Q3 2024 and for Self-hosted environments in Q1 2025) or the legacy Federation service that existed earlier.

# JFrog Artifactory Documentation

## Displayed in the header

### 6.6.7.1 | Monitor Federated Repositories using the Dashboard

Administrators can use the Federation dashboard to track the current state of your Federated repositories.

#### Note

The Federation dashboard requires an Artifactory release that supports the standalone Artifactory Federation service (release 7.104.2 or later).

#### At the site level, administrators can use the dashboard to:

- Understand the general health of all repository Federations at a glance
- Assign priority to selected repositories to provide them with greater access to system resources
- Filter the display to view all Federated repositories experiencing errors or delays

#### At the Federation level, administrators can use the dashboard to:

- Drill down to view the connection states between all members of a selected Federation
- View additional information about repository Federations, such as the number of binary tasks in progress

For more information about the Federation dashboard, see:

- [View the Status of All Repository Federations](#)
- [View the Status of a Selected Repository Federation](#)

### 6.6.7.1.1 | View the Status of All Repository Federations

Use the Federations dashboard to view a list of repository Federations and their current synchronization status.

#### To view the repository Federation status table:

In the Administration module, select Monitoring > Federation Status.

#### View Status Summaries

The tiles displayed above the table indicate how many repository Federations on this JPD are currently in the following statuses:

Status	Description
Healthy	Federations can attain a Healthy status only when <i>all</i> members are experiencing timely synchronization of metadata events.  For example, a Federation cannot attain this status if one of its members has an error, is involved with a Full Sync operation, or is experiencing delays.
Delayed	Indicates how many Federations have at least one member with high latency, which means that synchronization of metadata events is taking longer than expected.
Error	Indicates how many Federations have at least one member with synchronization errors.

# JFrog Artifactory Documentation

## Displayed in the header

Status	Description
Pending Full Sync	Indicates how many Federations have at least one member waiting for a Full Sync to be performed. A Full Sync is performed during the initialization process when a Federation is created, and as a way to restore synchronization when regular synchronization and auto-healing are insufficient (for example, if there are purged events that are no longer available to be synchronized).
Full Sync Running	Indicates how many Federations have at least one member currently undergoing the Full Sync procedure.

The state of each member is based on an aggregation of all connections that are federated with the JPD on which this endpoint is run. The aggregation is based on the most severe state. In the example shown below, metadata events sent from the Federated repository **gen-fed-fr** are synchronizing in a timely fashion with **gen-fed-au**. Therefore, that connection is **HEALTHY**. However, there is a delay in sending events from **gen-fed-fr** to **gen-fed-us**. Therefore, the aggregated status of **gen-fed-fr** is **DELAYED**.

### Filter the Repository Federations Table

Click one of the status cards to filter the Repository Federations table according to that status. For example, clicking the **Healthy** card shows only those Federations whose members are all synchronizing events in a timely fashion.

**Tip**

Click the filter icon  above the table to filter it according to multiple statuses.

#### View the Status of Each Repository Federation

The Repository Federations table includes the following information:

Column	Description
Priority	A row displaying the flag icon indicates this Federated repository has been assigned priority to system resources. For more information, see <a href="#">Assign Priority to Federated Repositories</a> .
Repository Key	The repository key of the member of the repository Federation on this JPD.
Members	The number of members in this repository Federation. For more information, see <a href="#">View Federation Members</a> .
Status	<p>The overall status of the repository Federation. The status is based on the most severe status among its members.</p> <p>For example, if the Federation has 5 members, some of which are in Error status, the overall status of the Federation is shown as Error together with an indication of how many members have that status.</p> <p><b>Note</b></p> <p>The status severity order (from highest to lowest) is as follows:</p> <ul style="list-style-type: none"><li>• Error</li><li>• Delayed</li><li>• Pending Full Sync</li><li>• Full Sync Running</li><li>• Disabled (the user has removed the member from the Federation)</li><li>• Unsupported (the member is running a version of Artifactory that does not support the Federation dashboard)</li><li>• Healthy</li></ul>

### View Federation Members

In one of the table rows, click the rectangle displaying the number of Federation members to open a popup window with a list of the members and their current status.

6.6.7.1.1.1 | Assign Priority to Federated Repositories

When you assign priority, you provide the event queue of the selected Federated repository with priority access to system resources. This is done to help ensure the timely synchronization of its events with the other members of the Federation.

#### Note

Since system resources are finite, priority should be assigned sparingly, as prioritizing too many repositories effectively cancels out the benefits of prioritization. The limit is 10 repositories or 20% of the total number of repositories, whichever is lower.

#### To assign priority:

1. Go to the table row for the repository you want to prioritize and hover over the **Priority** field. A flag icon will appear.
2. Click the flag icon to assign priority to the selected repository.

3. To remove priority from a selected repository, click the flag icon to remove it from the table.

#### Note

To prioritize an entire Federation you must assign priority to each member of the Federation separately.

# JFrog Artifactory Documentation

## Displayed in the header

6.6.7.1.2 | View the Status of a Selected Repository Federation

Select a Federated repository from the **Federations** table to view the status of the connections between this repository and the other Federation members, including the current latency and the number of queue events waiting to be sent.

To view the status of a selected repository Federation:

1. In the **Administration** module, select **Monitoring > Federation Status** to open the **Federations** page.
2. Click the name of the repository key belonging to the Federation you wish to examine. The **Federation Status** window is displayed.

The top of the **Federated Status** window contains the name of the selected Federation member and the following information about it:

- The number of replication tasks that are currently in progress.
- The total number of replication tasks that have failed to complete.

The **Metadata Event Connections** table includes a row for each incoming and outgoing connection related to the selected member. The table contains the following information about each connection:

Column	Description
Source	The repository on the source member responsible for sending events.
Target	The repository on the target member responsible for receiving events.
Status	The current status of the connection between the target and source members:

# JFrog Artifactory Documentation

## Displayed in the header

Column	Description
	<ul style="list-style-type: none"><li><b>Healthy:</b> Indicates there is timely synchronization of all metadata events from the source to the target.</li><li><b>Delayed:</b> Indicates whether the synchronization of metadata events between the source and target is taking longer than expected.</li><li><b>Error:</b> Indicates at least one synchronization error between the source and target.</li><li><b>Pending Full Sync:</b> Indicates that a Full Sync will soon be performed between the source and target.</li><li><b>Full Sync Running:</b> Indicates that the connection between the source and target is undergoing the Full Sync procedure.</li><li><b>Disabled:</b> Indicates the target is disabled, which means no events are being sent to it.</li><li><b>Unsupported:</b> Indicates the target is running a legacy version that does not support the Federation dashboard.</li><li><b>N/A:</b> Indicates that the status cannot be determined because the source or target is unreachable.</li></ul>
Latency	The length of time that has passed since the last event that was handled successfully.
Queued Events	The total number of events waiting to be sent to the target repository.

### Tip

To view the connections between a different source member and the other members of the Federation, select the source from the dropdown list. An icon  is displayed next to the selected member for easier identification. Click the x icon to view the connections between all members.

#### 6.6.7.2 | Monitor Federated Repositories using REST APIs

You can monitor the status of Federated repositories using a set of dedicated REST APIs. The endpoints that are available are dependent on whether you are using the standalone Artifactory Federation Service (introduced for Cloud environments in Q3 2024 and for Self-hosted environments in Q1 2025) or the legacy Federation service that existed earlier.

##### Monitoring APIs for the Standalone Artifactory Federation Service

The Artifactory Federation Service includes the following monitoring APIs:

- Get Federation Connection Details
- Get Federation State Details
- Get Federation State Summaries
- Prioritize Federated Repository
- Reconnect to Federation Client

##### Monitoring APIs for the Legacy Federation Service

The legacy Federation service includes the following monitoring APIs:

- Get Federated Repository Status
- Get Federated Repository Status (v2)
- Get Federation Sync State
- Get Federation Mirror Lag Time
- Get Unavailable Mirrors

#### 6.6.7.3 | Monitor Federated Repositories using Open Metrics

In addition to viewing the status of Federated repositories within the platform UI, it is possible to gather Federation performance data using a set of metrics that complies with the Open Metrics standard. This data can then be accessed by 3rd party monitoring tools without the need for custom inbound API calls.

### Note

This feature was introduced in version 7.63.7 and is available for Self-hosted customers only.

The Open Metrics that are available is dependent on whether you are using the standalone Artifactory Federation Service (introduced for Cloud environments in Q3 2024 and for Self-hosted environments in Q1 2025) or the legacy Federation service that existed earlier.

# JFrog Artifactory Documentation

## Displayed in the header

### Open Metrics for the Artifactory Federation Service

When using the Artifactory Federation Service, see [Artifactory Federation Service Custom Metrics](#).

### Open Metrics for Legacy Federation

When using the legacy Federation service, see the following topics:

- Setup: [Artifactory System Properties for Federated Repository Metrics](#)
- How to use: [Federated Repository Metrics](#)
- How to use in MBeans: [Federated Repository Metrics in MBeans](#)

6.6.7.3.1 | [Artifactory Federation Service Custom Metrics](#)

This section outlines the custom metrics collected by the standalone Artifactory Federation service.

#### Metrics Granularity

Due to potential performance overhead, custom metrics are collected by default at the tenant level. However, selected metrics can collect data at the repo key or member level if predefined thresholds are exceeded.

#### Warning

These granular metrics should be implemented and used with caution due to their potential performance impact.

#### Available Metrics

Custom metrics for the Artifactory Federation service are divided into the following categories:

- Lag Metrics
- Event Metrics
- Full Sync Metrics
- Member State Metrics
- Miscellaneous Metrics

#### Lag Metrics

The Artifactory Federation service includes the lag metrics described in the table below.

Name	Description	Type	Granularity
jfrtfss_worker_lag_seconds	<p>The lag time when transferring events to the remote member.</p> <p>The calculation:</p> <p>Lag = Current time - Trailing baseline time</p> <p>Trailing baseline time =</p> <ul style="list-style-type: none"><li>• Last processed event time</li><li>or</li><li>• Trailing stream detection time (when the stream takes longer than the grace period to process the events it receives)</li></ul>	Gauge	Per tenant
pq_queue_lag	The lag of the Artifactory Federation queue (in milliseconds). {queue_name="jfrog_federation_service_queue"}	Gauge	Per tenant
pq_queue_events_left	The number of events that remain to be handled to overcome the lag. {queue_name="jfrog_federation_service_queue"}	Counter	Per tenant

#### Event Metrics

The Artifactory Federation service includes the event metrics described in the table below.

# JFrog Artifactory Documentation

## Displayed in the header

Name	Description	Type	Granularity
jfrtfs_events_received_from_artifactory	The total number of events received by the Federation service from Artifactory.	Counter	Per tenant
jfrtfs_events_removed_in_filter_total	The total number of deduped (deduplicated) events filtered out of the queue.  For example, imagine an artifact with two events in the queue. In the first event, the artifact is updated and in the second it is deleted. The first event is removed from the queue to streamline the process, and only the deletion event is sent to the remote member.	Counter	Per tenant
jfrtfs_events_received_from_remote_rtfs	The total number of events received by the tenant from remote members.	Counter	Per tenant
jfrtfs_events_sent_to_remote_instance_total	The total number of events sent from the tenant to remote members.  <b>Note</b>  This metric includes both members running the Artifactory Federation service and members running the legacy Federation service.	Counter	Per tenant
jfrtfs_repo_management_message_blob_duration	Tracks the duration between the creation of partitions in the message_blob table in the RTFS DB.	Gauge	Per Tenant
jfrtfs_repo_management_events_duration	Tracks the duration between the creation of partitions in the events table in the RTFS DB.	Gauge	Per Tenant

### Full Sync Metrics

The Artifactory Federation service includes the Full Sync metrics described in the table below.

Name	Description	Type	Granularity
jfrtfs_full_sync_e2e_member_execution_count	Counts the number of Full Sync operations that have been executed.	Counter	Per Tenant / Member
jfrtfs_full_sync_e2e_member_execution_time_threshold	Counts the end-to-end duration of all Full Sync operations.  Configure the threshold using the property: <code>rfts.full.sync.execution.time.threshold.min</code> (default=30 min.)	Timer	Per Tenant / Member (per member if a single Full Sync operation exceeds the threshold)
jfrtfs_full_sync_sort_rate_artifact_per_second	The file list sort rate (in milliseconds).  The rate is calculated by dividing the number of artifacts in the file list by the total sort time.  Configure the threshold using the property: <code>rfts.file.list.sort.rate.threshold.ms.per.artifact</code> (default = 10000 ms)	Gauge	Global
jfrtfs_full_sync_artifact_events_propagation_rate	The artifact event propagation rate in milliseconds.  The rate is calculated by dividing the number of artifact events sent to the remote	Gauge	Global

# JFrog Artifactory Documentation

## Displayed in the header

Name	Description	Type	Granularity
jfrtfs_full_sync_file_list_artifacts	<p>member by the total amount of time needed to propagate them (i.e. elapsed time between sending events and receiving an OK response from the remote member).</p> <p>The total number of artifacts in the file list of the local member.</p> <p>Configure the threshold using the property:  <code>rtfs.file.list.size.metrics.threshold</code>          (default = 1 million)</p>	Counter	Per Tenant / Member (if the threshold is exceeded)
jfrtfs_full_sync_file_list_artifacts_rate	<p>the file list compilation rate in milliseconds.</p> <p>The rate is calculated by dividing the number of artifacts in the file list by the total amount of time to get the file list.</p> <p>Configure the threshold using the property:  <code>rtfs.file.list.process.rate.threshold.ms.per.artifact</code>          (default = 5 sec.)</p>	Gauge	Global

### Member State Metrics

The Artifactory Federation service includes the member state metrics described in the table below.

Name	Description	Type	Granularity
jfrtfs_auto_healing_recovered_members	The total number of recovery operations that have been performed using the auto-healing process. For example, recovery performed once on 5 members is equivalent to recovery being performed on the same member 5 times.	Counter	Global
jfrtfs_members_in_status	<p>The current number of members in the following states:</p> <ul style="list-style-type: none"> <li>• FS_RUNNING</li> <li>• PENDING_FS</li> <li>• IMPORT_IN_PROGRESS</li> <li>• ERROR_GENERIC</li> <li>• ERROR_OUT_OF_SYNC</li> <li>• ERROR_INACTIVE</li> </ul> <p>Disabled members are filtered out.</p>	Gauge	Per Tenant
jfrtfs_disabled_members	The total number of members disabled locally by the user.	Counter	Per Tenant

### Miscellaneous Metrics

The Artifactory Federation service includes the additional metrics described in the table below.

Name	Description	Type	Granularity
jfrt_rpc_response_from_rtfs_total {statusCode!=200}	The total number of HTTP errors between Artifactory and the Artifactory Federation service.	Counter	Global
jfrtfs_db_failed_query_count_total	The total number of failed database queries in the Artifactory Federation service.	Counter	Global
jfrtfs_db_failed_query_count_metric	The total number of failed database queries.	Counter	Global
jfrtfs_db_query_duration_metric	The amount of time that was needed to execute a specific query.	Counter	Depends on query

# JFrog Artifactory Documentation

## Displayed in the header

Name	Description	Type	Granularity
jfrtfs_http_client_available	The percentage of available HTTP client connections.	Gauge	Global
jfrtfs_http_client_leased	The percentage of leased HTTP client connections.	Gauge	Global
jfrtfs_http_client_pending	The number of pending HTTP client connections.	Counter	Global

### 6.6.7.3.2 | Artifactory System Properties for Federated Repository Metrics

The following Artifactory system properties are used to configure Federated repository monitoring data based on the Open Metrics standard:

- `artifactory.federated.metrics.enabled`

Determines whether metrics writing for Federated repositories is enabled (true/false). The default value is **false**.

- `artifactory.federated.metrics.sync.threshold.time.ms`

Defines the minimum threshold (in ms) above which the Open Metrics counter will indicate a repository has breached the sync threshold. The default value is **20000** (20 seconds).

#### Note

The default value is also the minimum value.

- `artifactory.federated.metrics.monitoring.interval.time.sec`

Defines the measurement interval for the Federation synchronization of all repositories. The default value is **30 seconds**.

#### Note

The system will override any defined value that is under the minimum set by JFrog to protect the system, currently set for 5 seconds.

- `artifactory.federated.metrics.exclude.disabled`

Determines whether to exclude disabled mirrors (whose sync lag by definition is always increasing) from metrics writing. The default value is **true**.

- `artifactory.federated.metrics.enable.count.disabled.members`

Determines whether to count the number of disabled members across all existing Federations in the JPD. The default value is **true**.

### 6.6.7.3.3 | Federated Repository Metrics

The metrics available for monitoring Federated repositories are based on the Get Federation Mirror Lag Time REST API:

```
GET api/federation/status/mirrorsLag
```

#### Important

The first four metrics described below are applicable only to healthy Federations. These metrics ignore Federated repositories that are down or disabled.

#### Federation Mirror Lag

Example:

```
jfrt_federation_member_lag_total{local_repo_key="example-repo-local", remote_repo_key="test3-repo", remote_url="http://docker.for.mac.localhost:10102/artifactory/"} 5000  
1575374691025
```

Records all Federated repositories whose sync lag exceeds the defined threshold (in milliseconds).

#### Note

The measured lag must exceed the minimum threshold value defined in the system property `artifactory.federated.metrics.sync.threshold.time.ms`.

#### Federation Mirror Maximum Lag

Example:

```
jfrt_federation_max_lag_ms_total 500 1575374698000
```

Records the maximum lag value (500 in the example above) from among all Federated repositories in milliseconds.

#### Number of Mirrors Exceeding the Sync Threshold Value

Example:

```
jfrt_federation_num_mirrors_exceeding_lag_threshold_total 17 1575374698000
```

Records the total number of mirrors (17 in the example above) that have a sync lag larger than the default threshold. If no mirrors have a log above the threshold, the recorded value will be 0.

#### Number of Federated Repositories Exceeding the Sync Threshold Value

Example:

# JFrog Artifactory Documentation Displayed in the header

```
jfrt_federation_num_repos_exceeding_lag_threshold_total 9 1575374698000
```

Records the total number of distinct Federated repositories (9 in the example above) that have a sync lag larger than the default threshold. This value will always be equal to or less than the Number of Mirrors Exceeding the Sync Threshold value. If no repositories have a log above the threshold, the recorded value will be 0.

## Number of Disabled Federated Members

Example:

```
jfrt_federation_num_members_disabled_total 2 1575374698000
```

Records the number of disabled mirrors (2 in the example above), including disabled Federated repositories from the JPD in context as well as disabled remote Federated repositories that are part of any existing Federation from the JPD in context.

## Number of Federated Repositories per Status

Examples:

```
jfrt_federation_num_repos_status_total{status="healthy"} 232 1575374691025
jfrt_federation_num_repos_status_total{status="out_of_sync"} 22 1575374691025
jfrt_federation_num_repos_status_total{status="error_exhausted"} 13 1575374691025
jfrt_federation_num_repos_status_total{status="pending_fullsync"} 41 1575374691025
```

Records the number of Federated repositories that have the indicated status. Possible statuses include: healthy, pending full sync, full sync running, out of sync, exhausted, disabled.

### 6.6.7.3.4 | Federated Repository Metrics in MBeans

An MBean section for Federated repositories is available whose attributes are identical to the metrics specified in Federated Repository Metrics. The attributes include:

- MaximumLagInMs
- NumberOfDisabledFederationMembers
- NumberOfReposExceedingLagThreshold

This metric requires the use of the Get Federation Mirror Lag Time API to limit the results to those repositories that exceed a defined threshold.

- NumberOfMirrorsExceedingLagThreshold

### 6.6.7.4 | Monitor Federated Repositories in the Platform UI

Monitoring Federated repositories in the platform UI is dependent on whether you are using the standalone [Artifactory Federation Service](#) (introduced for Cloud environments in Q3 2024 and for Self-hosted environments in Q1 2025) or the legacy Federation service that existed earlier.

#### Monitoring with the Artifactory Federation Service

When using the Artifactory Federation Service, see [Monitor Federated Repositories using the Dashboard](#).

#### Monitoring with Legacy Federation

When using the legacy Federation service, see the following topics:

# JFrog Artifactory Documentation

## Displayed in the header

- [View Federated Repository Status](#)
- [View Federation Sync Status](#)

6.6.7.4.1 | [View Federated Repository Status](#)

From Artifactory version 7.58.0, administrators can view the current status of a Federated repository, including the number of pending events, the mirror latency, and the synchronization status among the mirrors for this repository.

To view the status of a Federated repository:

1. Go to the **Administration** module and select **Repositories**.
2. Select the **Federated** tab.
3. Click the actions menu for the relevant Federated repository and select **Federation Status**.



The top of the Federated Repository Status window displays the following information about the selected JPD:

Item	Description
Replicated Artifacts	<b>Fully:</b> The total number of artifacts that have been fully replicated (metadata & binary) on this JPD. <b>Metadata only:</b> The total number of artifacts for which metadata exists on this JPD but not the binary.
Binaries Tasks	<b>In Progress:</b> The number of replication tasks that are currently in progress. If this value is significantly less than the Metadata Only value, this indicates a high failure rate that should be investigated. <b>Failed:</b> The total number of replication tasks that have failed to complete.

The table in the Federated Repository Status window displays the following information about each target mirror that is Federated with the selected repository:

Item	Description
Queue Events	The total number of events waiting to be sent to the target mirror.
Error Events	The number of failed events during the synchronization process.
Mirror Latency	The total wait time for the next pending event that needs to be handled. The calculation is: Current time – Time of the next event that has yet to be handled
Sync Status	Indicates whether the Federated repository is synchronized with the target mirror. (The threshold is a certain number of error events that is defined at the system level and cannot be modified.)

**View Queued Event Details for the Selected Target Mirror**

Select a row in the Federated Repository Status table to display additional details about the queue events for the selected target mirror.

Item	Description
Create	The number of Create events waiting to be sent to the target mirror.
Update	The number of Update events waiting to be sent to the target mirror.
Delete	The number of Delete events waiting to be sent to the target mirror.

# JFrog Artifactory Documentation

## Displayed in the header

Item	Description
Properties	The number of node properties waiting to be sent to the target mirror.

6.6.7.4.2 | [View Federation Sync Status](#)

From Artifactory version 7.55.1, administrators can see whether there are significant synchronization delays between the Federated repositories on the local JPD and other Federation members on remote JPDs.

To view the Federation synchronization status, go to the **Administration** module and select **Monitoring > Federation Status**.

The table contains the following information about each JPD:

Column	Description
URL	The URL of the JPD.
Binaries to Fetch	The total number of binaries that this JPD needs to fetch from Federated repositories on remote JPDs.
Fetch Failures	The number of fetch actions that have failed after a predefined number of retries.
Sync Status	Indicates whether a Federation repository on the JPD has exceeded the synchronization delay threshold with at least one other Federation member.  For example, when a new artifact is deployed to the JPD, the JPD must send the metadata about this event to the other Federation members. If there is a delay in this process that exceeds a defined period of time, the Sync Status becomes Delayed.  There are two possible sync statuses for the JPD - Federated and Delayed. The default threshold value is 50% of the value defined by the system property <code>events.log.cleanup.age.of.entries.to.discard.days</code> .  For example, if the system property for discarding log events is set to 3 days, the JPD sync status becomes Delayed if there is event metadata that has not been sent after a delay of 36

## JFrog Artifactory Documentation Displayed in the header

Column	Description
Most Delayed	Shows the length of the longest synchronization delay between a Federated repository in the JPD and another Federation member, including the name of the relevant repository.  hours.

### Refresh the Federated Sync Status

In the Federation Sync Status window, click **Refresh** to update the information in the table.

### View the List of Longest Delays in a JPD

In the Federation Sync Status table, click the row of a JPD to view a popup window containing a list of repositories (up to 10) with the longest delays.

The table contains the following information:

Column	Description
Source	The Federated repository on the source JPD.
Target	The Federated repository on the target JPD.
Delay	The length of the delay between the source to the target.

### 6.6.8 | Multi-Version Support

Artifactory includes multi-version support, which enables the members of a Federation to run different versions of Artifactory, even if the version at one site includes configuration features and values that are not supported on the versions running at other sites. Thanks to multi-version support, future upgrades after 7.49.6 can be performed on one site at a time, eliminating the need for simultaneous upgrades across all locations.

# JFrog Artifactory Documentation

## Displayed in the header

### Note

It is a prerequisite of the multi-version support feature to upgrade all Artifactory instances hosting Federated repository members to Artifactory 7.49.6. After this has been done, multi-version support is enabled for all versions going forward.

Whenever an instance with a new Artifactory version is introduced to the Federation, any configuration differences that are detected between the new version and older versions are reported to the user. For example, if there are new features not found in the older version, a message such as the following is displayed:



In the example shown above, the user can respond by either disabling the two new fields in the Federation member running the new Artifactory version, or by removing the member from the Federation.

If an existing feature has been enhanced with a new value that is not available in older versions, a message such as the following is displayed:



In such cases, the user can respond by selecting a value that the older versions support.

When creating repositories using the API, the process works in a more automated fashion to prevent other automated processes (such as disaster recovery) from breaking due to the Federation member that was upgraded.

If there are new features that older versions do not support, the new feature is disabled. For upgraded features, a default value is chosen that is supported on all member versions.

## 6.7 | Release Bundle Repositories

### Note

This feature is relevant for Enterprise+ customers who use JFrog Distribution to distribute Release Bundles v1. For information about Release Bundles v2, see [Release Lifecycle Management](#).

The Release Bundle repository protects your artifacts as part of the distribution flow. Artifacts that have been created and signed are automatically copied and saved into this separate repository where their contents cannot be edited or removed. This ensures consistency of distribution among target instances. Even if the original artifacts are removed from the original repository, they will continue to stay in the release bundle repository, available for distribution. These Release Bundles are created and managed in the Distributing Release Bundles page, and generally distributed from a source Artifactory instance to JFrog Artifactory Edge nodes.

Release bundles are stored with the following naming convention: {bundle name}/{bundle version}/{target path}

### 6.7.1 | Create a Release Bundle v1 Repository

The *release-bundles* repository is automatically created and used by default. However, users can create additional repositories with their own naming conventions that can also be used for distributing release bundles (v1).

### Note

The application supports only default repositories.

#### To create a Release Bundle v1 repository:

1. In the **Administration** module, select **Repositories**.
2. Click **Create a Repository** and select **Release Bundle** from the list.

### Common Basic Settings

The following settings are common to all package types. For a description of each setting, go [here](#).

- Repository Key
- Public Description
- Internal Notes
- Include and Exclude Patterns

### 6.8 | Repository Replication

Artifactory supports 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
- Overcoming 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 performing replication between two servers running the same version of Artifactory. If one of the two servers has been upgraded to a newer version, replication can typically continue without issues, but it is recommended to upgrade the other server to the same version as soon as possible.

#### Warning

We do not recommend using Artifactory repository replication in conjunction with AWS S3 cross-region replication of your filestore. Such a configuration can cause synchronization issues.

#### Note

[Learn more](#) about how to tune Cron Replication for a large number of artifacts.

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.

#### 6.8.1 | 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 near-real-time since each create, copy, move or delete of an artifact is immediately propagated to the far end.

#### Advantages of Push Replication

- 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) 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, the exponential growth of disk-space usage.

#### 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.

### 6.8.2 | Pull Replication

Pull replication provides a convenient way to populate a remote cache proactively, 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 having to wait for the file upload to be completed at the source.

#### Advantages of Pull Replication

- 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 be cached on-demand using the normal cache and proxy behavior of a Remote Repository.

### 6.8.3 | Schedule and Configure 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 service 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.

#### 6.8.3.1 | Configure Push Replication

A push replication task for a Local Repository is configured in the **Replication** tab of the Configuring a 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	Description
Destination URLs	The replication targets you have defined
Enabled	When set, enables replication of this repository to the target specified in <b>Push to</b>
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.

##### 6.8.3.2 | Add a Push Replication Target

To add a target site for this replication, click **Add** to display the **Replication Properties** dialog, and fill in the following details.

# JFrog Artifactory Documentation

## Displayed in the header

Field	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. Use the format <code>https://&lt;artifactory_url&gt;/artifactory/&lt;repository_name&gt;</code> .
Username	The HTTP authentication username.
Credentials	Use either the HTTP authentication password or identity token
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).  <b>Warning</b> Enabling this option will delete artifacts on the target that do not exist in the source repository. If the source repository is empty, replication will purge the contents of the target repository.
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.

### 6.8.3.3 | Configure Pull Replication

A pull replication task for a Remote Repository is configured in the **Replication** tab of the **Edit Remote Repositories** 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.

Field	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. Use the format <code>https://&lt;artifactory_url&gt;/artifactory/&lt;repository_name&gt;</code> .  For some package types, you need to prefix the repository key in the URL with <code>api/&lt;pkg&gt;</code> . For a list of package types where this is required, see the note below.
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.
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.

#### 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 [Configure a Smart Remote Repository](#).

#### 6.8.4 | Replicate with REST API

Both Push and Pull Replication are supported by Artifactory's REST API. For details please refer to the following:

# JFrog Artifactory Documentation

## Displayed in the header

- Get Repository Replication Configuration
- Set Repository Replication Configuration
- Update Repository Replication Configuration
- Delete Repository Replication Configuration
- Scheduled Replication Status
- Pull/Push Replication

### 6.8.5 | Replication Properties

Once replication has been invoked, the system 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 Tree 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: <b>ok:</b> The replication succeeded <b>failure:</b> The replication failed. You should check the log files for errors
artifactory.replication.<source_repo_key>.finished	Indicates when the replication finished

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: <b>ok:</b> The replication succeeded <b>failure:</b> 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

### 6.8.6 | Optimize Repository Replication Using Storage Level Synchronization Options

#### Note

Requires an Enterprise+ license.

You can set Artifactory to offload the heavy-lifting work of replicating data to the storage device, by only replicating the metadata while ensuring the data is available on the target binary store. This is recommended, for example, when you have two Artifactory instances configured with replication between them. The binary provider configured on Artifactory includes integrated support for replicating data on the storage level, allowing you to assign the replication process to the storage.

To run repository replication using storage level synchronization options:

# JFrog Artifactory Documentation

## Displayed in the header

- Synchronize the storage devices for the source and target Artifactory systems.
- Set the `checkBinaryExistenceInFilestore` flag to true in the Push or Pull Replication API commands in the source Artifactory. For more information, see the Pull/Push Replication, Set Repository Replication, and Update Repository Replication API commands.
- Set the `checkBinaryExistenceAllowed` flag to true in the target Artifactory with the `checksumReplication` API command. For more information, see Configure Checksum Replication and Get Checksum Replication API commands.

### Enable the Flag During Replication

When enabling the flag, during replication, Artifactory searches for the binary in the target Artifactory instance in the binary storage and if it exists, the source replicates only the metadata.

- It is the user's responsibility to replicate the data on the storage level.
- This feature is disabled by default and does not change any behavior.

## 6.9 | Repository Layouts

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, the system 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.

### 6.9.1 | The Freedom of Custom Layouts

Repository Layouts enable you to take full control over the layout used by each repository and use layout definitions to identify module artifacts and descriptors. Repository layouts support these smart module management facilities for any build technology:

- Automatic snapshot/integration version cleanup
- Deleting old versions
- Conversions between remote and local layouts
- Conversions between two 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)

### 6.9.2 | 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.

### 6.9.3 | Modules and Path Patterns used by Repository Layouts

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"	

# JFrog Artifactory Documentation

## Displayed in the header

Field	Description	Example	Mandatory
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"	✗

### 6.9.3.1 | 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.).

#### 6.9.3.1.1 | 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 separate path patterns for descriptor-type artifacts (such as a .pom or an ivy.xml file).

The following tokens are available:

Token	Description
[org]	Represents the <b>Organization</b> field where the levels are separated by dots ('.'), a-la Ivy. For example: "org.slf4j"
[orgPath]	Represents the <b>Organization</b> field where the levels are separated by path separators ('/'), a la Maven. For example: "org/slf4j"
[baseRev]	Represents the <b>Base Revision</b> field
[module]	Represents the <b>Module</b> field
[folderIntegRev]	Represents the <b>Folder Integration Revision</b> field
[fileIntegRev]	Represents the <b>File Integration Revision</b> field
[classifier]	Represents the <b>Classifier</b> field
[ext]	Represents the <b>Extension</b> field
[type]	Represents the <b>Type</b> field
[customTokenName<customTokenRegex>]	A custom token. Can be used to create a new type of token when the provided defaults are insufficient. For example, [myIntegRev<ITEG-(?:[0-9]+)>] creates a new custom token named myIntegRev that matches the word ITEG followed by a dash and a minimum of one digit.

# JFrog Artifactory Documentation

## Displayed in the header

### 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".

#### 6.9.3.1.2 | Artifact Path Patterns

An artifact path pattern represents the typical structure in which all module artifacts are expected to be stored.

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]
```

#### 6.9.3.1.3 | 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 path 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
```

#### 6.9.4 | Configure Repository Layouts

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.

For complete details, see:

- [Layout Configuration](#)
- [Repository Layout Configuration](#)

#### 6.9.4.1 | Layout Configuration

Layout configuration is available to administrator users in the **Administration** module under **Repositories | Layouts**.

Additional layouts can be created from scratch by clicking **New** or by duplicating an existing layout.

## Testing Layouts

Once you have finished configuring your layout, you can test it on an artifact path, and see how the system builds module information from the path, using the layout definitions.

### Path Patterns

Path patterns 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](-[folderIntegRev])/[module]-[baseRev](-[fileIntegRev])(-[classifier]).[ext]
```

6.9.4.1.1 | 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 ("2.3-2010108.100922-2"), 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}

### 6.9.4.2 | Repository Layout Configuration

#### Before custom layouts

Repositories created before the introduction of custom repository layouts are automatically configured with the default Maven 2 layout.

The following topics describe repository layout configuration:

- Local Repository Configuration
- Remote Repository Configuration
- Virtual Repository Configuration

### 6.9.4.2.1 | Local Repository Layout Configuration

Layouts are mandatory for local repositories since they define the structure with which artifacts are stored.

#### To select a repository layout for a local repository:

1. In the **Administration** module, select **Repositories**.
2. Do one of the following:
  - When creating a new repository, click **Create a Repository** and then select **Local**.  
or
  - When modifying an existing repository, click the **Local** tab, locate the relevant repository and click its repository key.
3. In the **Basic** tab, scroll down to the Repository Layout field. When you create a new repository, Artifactory will recommend the best layout according to the package type.

4. Select a repository layout from the list and then click **Save**.

#### 6.9.4.2.2 | Remote Repository Layout Configuration

Layouts are mandatory only for the remote repository cache configuration. However, you can also specify the layout of the remote repository itself, as described in the procedure below.

##### To configure a repository layout for a remote repository:

1. In the **Administration** module, select **Repositories**.
2. Do one of the following:
  - When creating a new repository, click **Create a Repository** and then select **Remote**.
  - or
  - When modifying an existing repository, click the **Remote** tab, locate the relevant repository and click its repository key.
3. In the **Basic** tab, scroll down to the Repository Layout field.

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, suppose you have a remote repository that 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.

4. Select a repository layout from the list and then click **Save**.

#### 6.9.4.2.3 | Virtual Repository Layout Configuration

You can 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, the system 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 is not configured with a layout

#### To configure a repository layout for a virtual repository:

1. In the **Administration** module, select **Repositories**.
2. Do one of the following:
  - When creating a new repository, click **Create a Repository** and then select **Virtual**.  
or
  - When modifying an existing repository, click the **Virtual** tab, locate the relevant repository and click its repository key.
3. In the **Basic** tab, scroll down to the Repository Layout field.
4. Select a repository layout from the list and then click **Save**.

## 6.10 | Repository Support for Package Clients

Use this table to identify the supported repository types for each of the package clients that Artifactory supports.

Package Client	Local Repositories	Remote Repositories	Virtual Repositories
Alpine	✓	✓	✓
Ansible	✓	✓	✓
Bower	✓	✓	✓

# JFrog Artifactory Documentation

## Displayed in the header

	Package Client	Local Repositories	Remote Repositories	Virtual Repositories
Cargo	✓	✓	✓	✗
Chef	✓	✓	✓	✓
CocoaPods	✓	✓	✓	✓
			Note	
			Remote repositories are only supported for CocoaPods when using CocoaPods CDN.	
Composer	✓	✓	✓	✓
Conan	✓	✓	✓	✓
Conda	✓	✓	✓	✓
CRAN	✓	✓	✓	✓
Debian	✓	✓	✓	✓
Docker	✓	✓	✓	✓
Generic	✓	✓	✓	✓
Git LFS	✓	✓	✓	✓
Go	✓	✓	✓	✓
Gradle	✓	✓	✓	✓
Helm	✓	✓	✓	✓
Hugging Face	✓	✓	✓	✗
Ivy	✓	✓	✓	✓
Maven	✓	✓	✓	✓
npm	✓	✓	✓	✓
NuGet	✓	✓	✓	✓
OCI	✓	✓	✓	✓
Opkg	✓	✓	✓	✗
P2	✗	✓	✓	✓
Pub	✓	✓	✓	✓
Puppet	✓	✓	✓	✓

	Package Client	Local Repositories	Remote Repositories	Virtual Repositories
PyPI	✓	✓	✓	✓
RPM/Yum	✓	✓	✓	✓
RubyGems	✓	✓	✓	✓
SBT	✓	✓	✓	✓
Swift	✓	✓	✓	✓
Terraform Registry	✓	✓	✓	✓
Terraform Backend Repository	✓		✗	✗
Vagrant	✓		✗	✗
VCS	✗		✓	✗

## 7 | Package Management

The JFrog Platform 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.

The Packages view in the Application module provides easy access to information about all the packages in your repositories and supports:

- Filtering the Packages List
- Viewing Package Information
- Viewing Xray Data on Packages
- Viewing Package Version Information
- Downloading Package Versions
- Adding Packages to Projects

To learn which repository types are supported for the different supported package types, see [Repository Support for Package Clients](#).

### 7.1 | Viewing Packages

The Packages list provides easy access to information about all the packages stored in all repositories in Artifactory. You have quick access to the most important summary information about the latest package versions and you can easily drill down for more details about previous versions. From the Packages list, you can create filters to focus on the packages that interest you according to a broad range of criteria, create customized views of filters for reuse and later reference, and view a list of the most recently viewed packages.

For some package types, you can download packages and copy installation commands when drilling down into a package.

To display the Packages list, from the **Application** module, go to **Artifactory| Packages**.

The Packages list is shown below. When you initially display the Packages view, **Recently Viewed** and **All Packages** tabs appear at the top left of the screen, and filter options appear on the right of the screen.

When you initially display the Packages list, all packages stored on Artifactory are listed, sorted lexicographically.

- For instructions on how to filter the packages list, click [here](#).
- For instructions on how to create customized views of package filters, click [here](#).

#### 7.1.1 | Filtering the Packages List

The initial Packages list displays all packages stored on Artifactory. To make use of this list, you need to use the filtering option to focus on the packages that you want to examine. You can use the filtering option to filter packages based on package types, package names, and other criteria to display only the packages that you want to view.

##### To filter the packages list:

1. In the All Packages list, go to the Filters section on the right side of the screen.

# JFrog Artifactory Documentation

## Displayed in the header

2. Filter the Packages list according to the following:

- **Package Type:** You can choose multiple package types. For example, if you choose Docker and Maven, the filter will list all Docker and Maven packages stored on Artifactory.
- **Package Name:** Enter a full or partial name. Wildcard search is applied to both the prefix and suffix of the text you enter. For example, if you enter **thon** and there is a package with the full name is **Python**, the filter will list the package.
- **Description:** The filter will search for the text that you enter here in the metadata descriptions of all packages and will list those packages where the text is found in any part of the metadata description.
- **Keywords:** If you are looking for packages that have keywords, enter the keyword here. Only one keyword is allowed.
- **Created On:** Choose from one of the following:

### Note

This filter applies only to the creation date of the first version of the package.

- **Last 7 Days:** The filter will list packages created in the last 7 days.
- **Last 30 Days:** The filter will list packages created in the last 30 days.
- **Last 90 Days:** The filter will list packages created in the last 90 days.
- **Custom:** The filter will list packages created between the start and end dates that you choose here.

### Note

When you apply any of the filter options above, a green dot appears by the option to indicate that this option is applied.

## Saving Filters for Reuse

Once you have filtered the Packages list to display the packages you want to review, you can save the filtered list and come back to it at a later point in time.

### To save a filtered list of packages:

1. At the bottom of the Filters section, click **Save New View**.

The View Name prompt appears.

2. In the View Name prompt, enter a descriptive name for the view and click **Save**. A short name is recommended since the name space in the UI is limited.

The view is saved, and appears at the top of the Packages list next to the Recently Viewed and All Packages tabs.

Once you have saved a view, there are further actions you can perform on it such as renaming the view, deleting it, or making it your default view. See the following section for instructions on managing Custom Views.

### Note

- You can create up to 8 Custom Views.
- Once you have created a Custom View, you can update it by displaying the view and clicking **Update Custom View** in the Filters section.
- Clicking on a package in the list to view its details will cause that package to be added to the Recently Viewed list, described below.
- Custom Views can only be created from the All Packages tab.
- Custom Views are saved on the browser cache level.

## Managing Custom Views

Once you have created a Custom View, there are further actions you can perform on it such as renaming the view, deleting it, or making it your default view.

### To perform an action on a Custom View:

1. Hover over the Custom View with your mouse so that 3 vertical dots appear to the right of the view name.

2. Click the 3 dots.

A drop-down menu appears.

3. From the drop-down, choose the action that you want to perform on the view.

# JFrog Artifactory Documentation

## Displayed in the header

### Recently Viewed

After you have started to use the Packages list and display information on specific packages, you can view those packages again by clicking **Recently Viewed** at the top left of the screen, which displays up to the 10 most recently viewed packages.

### 7.1.2 | Viewing Package Information

In the Packages list, the package summary information is displayed, with the package name and logo in the left top corner, and the creation date of the latest version and its version number. The following information is displayed in the upper right of the panel.

Field	Description
License	Name of the license covering the package
Versions	Number of versions of the package
Xray	Indicates the status of the Xray scan. For more information, see <a href="#">Xray Security and Compliance</a> . <i>Xray scanning requires Pro X, Enterprise with Xray, or an Enterprise+ license.</i>
Downloads	Total number of times the package (in its various versions) has been downloaded
Tags	Metadata tags (available only for npm and NuGet)

Click on a Package to view the Package versions.

In the **Versions** section, use the **View By** toggle to select one of the following views:

- **List:** Displays information about the package versions.
- **Graph:** Displays security and license violations informations from JFrog Xray with the number of downloads per version.

For more information, see [Xray Security and Compliance](#).

*Xray scanning requires Pro X, Enterprise with Xray, or an Enterprise+ license.*

The **List** option displays the following information about the package versions:

Field	Description
Version	Package version numbers

Field	Description
Repositories	Name of repositories that contain the package version
Digest	The package's SHA 256 digest (available only for Docker)
Last Modified	Date when the package version was last modified
Downloads	Number of times package version was downloaded
Xray Status	<p>The following Xray status indicators are displayed:</p> <ul style="list-style-type: none"><li>• Severity of the package vulnerability (Low / Medium / High)</li><li>• Not Scanned</li><li>• No Vulnerabilities</li><li>• Pending Scan</li></ul> <p>For more information, see <a href="#">Xray Security and Compliance</a>.</p> <p><i>Xray scanning requires Pro X, Enterprise with Xray, or an Enterprise+ license.</i></p>

#### NPM Packages Only

For npm package types, the  `npm i angular` appears to the right of the package name. For details, see [Adding Packages to Projects](#).

#### 7.1.3 | Viewing Xray Data on Packages

##### Note

Required JFrog Subscriptions



In the Package list view, you can quickly and periodically review the status of your security and compliance for all your scanned packages on your indexed resources to gain information about the Xray scan status and assigned licences on the latest version of the package.

From the list view, you can toggle to the Graph tab to view a graph displaying a breakdown according of security or license violations according to severity.

#### 7.1.4 | Viewing Package Version Information

Click on the version number to view details about a particular package version, in the detailed table.

The information in the summary section, in the top panel, now displays summary information about the selected package version.

To download the package version to your computer, click **Download**, located on the right, below the summary information. For more information, see [Downloading Package Versions](#).

The detailed table now appears with the following tabs and information:

##### Readme

Applies to npm packages. Contains readme documentation.

##### Builds

In the **Build** section, use the **View** toggle to select one of the following views:

- **Produced By:** Displays information about the builds that produced the package versions.
- **Used By:** Displays information about the builds that used the package versions as dependencies.

The information includes the name, number, and creation date of each build. Click on the build name to open the Build page with the full information about the build.

## Xray Data

*Xray scanning requires Pro X, Enterprise with Xray, or an Enterprise+ license.*

For more information, see [Viewing Xray data on Package Versions](#).

## Docker Layers

Applies to Docker packages. Lists the layer related information.

## Distribution

*Requires an Enterprise+ license.*

Displays the Release Bundles containing the package version, the Release Bundle Distribution status and when they were last updated. Click the Release Bundle Name to view the Bundle in the [Distribution](#) page.

## Repositories

Displays where the package versions exist in Artifactory. The locations are indicated by the repository names and the full paths to the packages in Artifactory. Enter version numbers or repository names to filter the list.

Click on the path to open the Artifact Repository Browser, showing the location of the package in the Tree view.

## Viewing Xray Data on Package Versions

Selecting a package version displays detailed Xray data information.

In the top pane, you can view the Xray severity and license assigned to the version.

Under the **Xray Data** tab, you can view these dedicated Xray related tabs with the option to run a set of actions on the version. For detailed information on each tab, see Analyzing Resource Scan Results.

Under the **Xray Data** tab, you can view these dedicated Xray related tabs with the option to run a set of actions on the version. For detailed information on each tab, see [Analyzing Resource Scan Results](#).

## 7.2 | Downloading Package Versions

This topic describes how to download a package version to your computer.

To download a package to your computer, follow these steps:

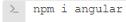
1. From the **Application** module, click **Artifactory**, and then click **Artifacts**.
2. Navigate to the desired repository and then locate the desired package version.
3. To download the package, do one of the following:
  - Right-click the package version, and then click **Download**.
  - Click the package, and then from the right corner click **Download** icon.

### Related Information

- To download via JFrog CLI, refer to the [Downloading Files](#).

## 7.3 | Adding Packages to Projects

It is usually more convenient to use the copy command button than using the **Download** button.

To add the latest version of package to a project, click  `npm i angular`. The command displayed in the text box is copied to the clipboard. Paste the command into the command line on your terminal. Execute the command line to automatically add the latest version of the package to the package.json file.

When the version-level information is displayed, select a specific version and click  `npm i angular@1.7.8` to copy the command for the selected version to the clipboard. Continue as described above to add the version of the package to the package.json file.

# JFrog Artifactory Documentation

## Displayed in the header

### 7.4 | Supported Package Types

The JFrog Platform supports the following package formats with new formats added regularly as the need arises.

Package Type	Description
Alpine Linux	Use Artifactory to gain full control of your deployment and resolution process of Alpine Linux (*.apk) packages.
Ansible	Ansible is an open-source platform that enables you to automate IT processes such as provisioning, application deployment, orchestration, and more.
Bower	Boost your front-end development by hosting your own Bower components and proxying the Bower registry in Artifactory.
Cargo	Enhance your capabilities for configuration management with Cargo using all the benefits of a repository manager.
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.
Conda	Artifactory natively supports Conda repositories for Python, R, Ruby, Lua, Scala, Java, JavaScript, C/C++, FORTRAN.
CRAN	Deploy and resolve CRAN packages for the R language using dedicated CRAN repositories.
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.
Generic	Define a repository as Generic to which you can upload packages of any type. Generic repositories are useful when you want to proxy unsupported package types, store installers, navigation files, audio files, etc.
Git LFS	Optimize your workflow when working with large media files and other binary resources.
Go Registry	Build Go projects while resolving dependencies through Artifactory, and then publish the resulting Go packages into a secure, private Go registry.
Gradle	Resolve dependencies from and deploy build output to Gradle repositories when running Gradle builds.
Helm (Helm OCI)	Manage your Helm Charts in Artifactory and gain control over deployments to your Kubernetes cluster.
Hex	Manage Hex packages in Artifactory for Elixir and Erlang projects.
Hugging Face Models & Data Sets	Create a single system of record for ML models that brings ML/AI development in line with your existing SSC, using Artifactory to integrate machine learning into your stack.
Machine Learning Repositories	Machine Learning Repositories with the FrogML SDK is a local management framework tailored for machine learning projects, serving as a central storage for models and artifacts, featuring a robust version control system. It offers local repositories and an SDK for effortless model deployment and resolution.
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 <a href="https://npmjs.org">npmjs.org</a> through Artifactory.
NuGet (.NET, Chocolatey, Powershell)	Host and proxy NuGet packages in Artifactory, and pull libraries from Artifactory into your various Visual Studio .NET applications.

Package Type	Description
NVIDIA NIM Models	Cache NVIDIA NIM models in Artifactory via the remote repository.
OCI	Manage OCI artifacts in Artifactory. OCI is an open-source container governance structure that sets clear industry standards for containers.
Opkg	Optimize your work with OpenWrt using Opkg repositories. Proxy the official OpenWrt repository and cache remote .ipk 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.
Pub /Dart Repositories	Artifactory natively supports Dart packages, giving you full control of your deployment and resolution process of Flutter, Angular Dart, and general Dart programs.
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 a 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.
Swift	Artifactory natively supports a dedicated Swift registry, giving you full control of the deployment and resolution process of your Swift packages and the dependencies.
Terraform/OpenTofu	A fully-fledged Terraform repository solution giving you full control of your deployment and resolve process of Terraform Modules, Providers, and Backend packages.
Vagrant	Securely host your Vagrant boxes in local repositories.
VCS	Consume source files packaged as binaries.

## 7.5 | Alpine Linux Repositories

Artifactory provides full support for managing Alpine Linux packages through local, remote, and virtual repositories. To learn more about Alpine, see [Alpine Linux Documentation](#).

Artifactory's support for Alpine Linux provides:

- The ability to provision Alpine Linux packages from Artifactory to the Alpine Linux command line tool from all repository types.
- Calculation of Metadata for Alpine Linux packages hosted in Artifactory's Local Repositories.
- Access to remote Alpine Linux registries (such as <https://pkgs.alpinelinux.org>) through Remote Repositories which provide proxy and caching functionality.
- The ability to access multiple Alpine Linux registries from a single URL by aggregating them under Virtual Repositories.
- Management of multiple RSA key pairs and enables you to set different RSA keys per Alpine repository, which allows you to sign Alpine Linux indexes in your Local and Virtual Repositories.

### Alpine Linux version support

Artifactory supports Alpine Linux version 3.9.6 and above and **apk-tools** client version 2.10.3 and above.

#### Note

Artifactory signs repository metadata (not packages) for Alpine.

### 7.5.1 | Alpine Linux Repository Structure

An Alpine Linux repository is a directory with a collection of **apk** files, and an Alpine Linux repository consists of the following three main coordinates:

- branch
- repository
- architecture

Artifactory uses the same convention of the directory layout as the Alpine Linux repository uses. For example:

```
|--- 3.9 (branch)
|   \--- main (repository)
|     \--- aarch64 (architecture)
|       \--- APKINDEX.tar.gz (index file)
|         \--- a2ps-4.14-r7.apk
|           \--- ...apk
|     \--- x86
|       \--- APKINDEX.tar.gz (index file)
|         \--- a2ps-4.14-r7.apk
|           \--- ...apk
|     \--- ...
|   \--- ...
```

### Deployment Structure

All deployment of Alpine Linux packages into Artifactory must be under the <BRANCH>/<REPOSITORY>/<ARCHITECTURE>/ structure.

If packages are not deployed under this structure, **they will not be included in any index file**.

#### 7.5.2 | Set up an Alpine Linux Repository

You can set up the following repository types:

- Local Repositories
- Remote Repositories
- Virtual Repositories

Follow the steps according to each repository type below. An Alpine Linux package (.apk) is deployed to a local Alpine Linux repository, and resolved using all repository types.

You can download packages from a local, remote, or virtual Alpine Linux repository.

### Prerequisites

Alpine Linux requires RSA keys by default. To add RSA keys, please refer to [RSA Key Pairs](#).

If you do not configure RSA keys, users have to use the `allow-untrusted` flag as described in [Resolving a Package](#).

Artifactory supports the signing of Alpine Linux index files, and not packages. To learn more about creating keys for Alpine Linux packages, click [here](#).

#### 7.5.2.1 | Local Alpine Repositories

Local repositories enable you to deploy Alpine Linux (.apk) packages. Artifactory calculates the metadata for all packages and indexes them to allow users to download these packages through the Alpine Linux client.

To create an Alpine Linux local repository, navigate to the **Administration** module, go to **Repositories** | **Repositories** | **Local** | **New Local Repository** and select **Alpine** as the **Package Type**.

In the **Advanced** tab, you can select an RSA key from the list to sign the Alpine Linux index file.

#### 7.5.2.2 | Remote Alpine Repositories

Remote Repositories enable you to proxy and cache Alpine Linux packages.

To specify that a Remote Repository supports Alpine Linux packages, you need to set its **Package Type** to **Alpine** when it is created.

#### Retrieve RSA keys in a Remote Repository

When setting up a Remote Repository, you will have to retrieve the RSA keys manually and set it in the apk client.

You can avoid retrieving the keys manually by aggregating a Remote Repository in a **Virtual Repository** that will enable you to use the Virtual Repository's key-pair to re-sign the aggregated index file.

In order to do so, configure your Virtual repository with a key-pair and then use Set Me Up to retrieve the Virtual public key to your local machine.

#### 7.5.2.3 | Virtual Alpine Repositories

A Virtual Repository defined in Artifactory aggregates packages from both local and remote repositories.

This allows you to access both locally hosted Alpine Linux packages and remote proxied Alpine Linux repositories from a single URL defined for the Virtual Repository.

To define a virtual Alpine Linux repository, do the following:

1. Create a **Virtual Repository**, and set the **Package Type** to **Alpine**.
2. Select the underlying local and remote Alpine Linux repositories to include in the **Basic** settings tab.
3. You can select an RSA key-pair from the list to sign the Alpine Linux packages.

#### RSA Key-pair

The RSA key-pair defined for the virtual repository will be used to sign the virtual index file. In case local repositories are defined with RSA key-pairs, these keys will be ignored.

#### 7.5.3 | Configure Alpine Linux Package Manager to work with Artifactory

In order to use Artifactory with your Alpine Linux client, first you need to set Artifactory as an Alpine Linux repository, and then add the relevant RSA public key to verify the index signature. Then you can proceed to resolve and deploy the relevant Alpine Linux package.

##### Step 1: Add Artifactory to your /etc/apk/repositories File

1. Navigate to **Application Module** | **Artifactory** | **Artifacts**.
2. Select the desired repository.
3. Select **Set Me Up**.
4. Copy the **General**'s section command and run it.

## Step 2: Verify the Index Signature

In order to verify index signature you need to add the repository RSA public key into your /etc/apk/keys folder.

Copy the **Set Me Up** public RSA key unique retrieval command and run it.

(note: the above image is an example only and you should not use it as-is. Use the **Set Me Up** dialog to get a unique command per your public key file name, as used to sign the index file)

If a local or virtual repository does not contain an RSA key-pair, you can either use the --allow-untrusted flag or request form your Admin to set a key-pair for the repository.

For signed indexes of remote repositories, please refer to Setting up a Remote Repository.

### 7.5.3.1 | Resolve an Alpine Package

To resolve an Alpine package, use the following cURL commands:

### 7.5.3.2 | Deploy an Alpine Package

To deploy an Alpine Linux package into an Artifactory repository you can use the following cURL with the relevant path parameters:

#### cURL

```
curl -H 'X-JFrog-Art-Api:<API_KEY>' -XPUT "https://localhost:8080/artifactory/alpine-local/<BRANCH>/<REPOSITORY>/<ARCHITECTURE>/<ALPINE_PACKAGE_NAME>" -T <PATH_TO_FILE>
```

Parameter	Example
branch	v3.9
repository	main
architecture	x86
alpine_package name_	grep-3.1-r2.apk

### Deploy an Alpine Package Using the UI

To deploy an Alpine Linux package to Artifactory, do the following:

1. Navigate to **Artifactory | Artifacts | Deploy**.
2. Select your Alpine Linux repository as the **Target Repository**.
3. In the **Target Path**, specify the relative path in the target repository:

<BRANCH>/<REPOSITORY>/<ARCHITECTURE>/<ALPINE\_PACKAGE\_NAME>

### Deployment Structure

For your files to be indexed properly, it is very important to ensure that all deployment of Alpine Linux packages into Artifactory occurs under the <BRANCH>/<REPOSITORY>/<ARCHITECTURE>/ structure. Packages deployed anywhere else will not be indexed.

### 7.5.4 | Alpine Artifact Metadata

Artifactory writes several entries from the Alpine Linux package's metadata as properties on all of the artifacts.

These properties can be used to search for Alpine Linux packages more efficiently using the **Artifacts Search**, by **Package type Alpine**.

Alpine Linux package properties are the following:

- alpine.name
- alpine.version
- alpine.branch
- alpine.repository
- alpine.architecture

### 7.5.5 | Alpine REST API Support

The Artifactory REST API enables the recalculation of the repository index, as described in Calculate Alpine Repository Metadata.

## 7.6 | Ansible Repositories

The JFrog Artifactory integration with Ansible allows you to manage Ansible collections in Artifactory. Ansible is an open-source platform that enables you to automate IT processes such as provisioning, application deployment, orchestration, and more. [Ansible Galaxy](#) is the public registry for Ansible roles and collections.

To learn more about collections and roles, refer to the following:

- Collections are a distribution format for Ansible content that can include playbooks, roles, modules, and plugins. You can install and use collections through a distribution server, such as Ansible Galaxy, or a Pulp 3 Galaxy server.
- Roles let you automatically load related vars, files, tasks, handlers, and other Ansible Artifacts based on a known file structure. After you group your content into roles, you can easily reuse and share it with other users.

### Main Features of Ansible in Artifactory

Artifactory supports managing Ansible roles and collections, ensuring optimal and reliable access to Ansible Galaxy.

- **Single Point of Truth**

This integration allows users to work with the Ansible registry, and manage and save their Ansible collections in Artifactory while providing full flexibility and usability.

- **Secure Private Ansible Registry with Fine-Grained Access Control**

Local Ansible repositories are where you store internal Ansible collections for distribution across your organization. With the fine-grained access control provided by built-in security features, Artifactory offers secure Ansible collections publish and install with local Ansible repositories as fully functional, secure, private Ansible registries.

- **Consistent and Reliable Access to Remote Roles and Collections**

Remote Ansible Repositories in Artifactory proxy external resources such as Ansible-Galaxy or a remote Ansible registry in another Artifactory instance, and cache downloaded images. This reduces overall networking and creates fast, consistent, and reliable access to roles and collections on these remote resources.

- Access multiple Ansible repositories from a single URL by aggregating them under a Virtual Repository. This overcomes the limitation of the Ansible client, which can only access a single registry at a time.
- Artifactory supports the installation of roles via Remote repositories if configured.

### Ansible Supported Clients

Ansible package registry supports the following and has been tested:

- Ansible Core version 2.17 or above
- Ansible community package release v10.x or above.

### Limitations of Ansible in Artifactory

The following are the limitations of Ansible in Artifactory:

- **Remote Role Installing:** Artifactory does not support installing roles from any Git repository via remote repositories.
- **Local Role Publishing:** Artifactory does not support publishing roles into local repositories. Instead, roles must be wrapped in collections. For instructions on how to do this, please see [Migrating roles to collections](#).
- **Repository Browsing:** Artifactory does not support browsing Ansible remote registries. The artifact tree only shows packages explicitly pulled into Artifactory, not the contents of the remote registry.

#### 7.6.1 | Get Started with Ansible

Get started with the Ansible registry and its workflow.

The following diagram illustrates the administrator user and application user activities:

Following are the modules you work with based on your role:

- Administrator Module - Ansible Packages
- Application Module - Ansible Packages

##### 7.6.1.1 | Administrator Module - Ansible Packages

The activities of the administrator user are outlined in the following table:

#	Task	Description	For UI, see...	For API, see...
1	Artifactory Login	Log into your Artifactory with the credentials	NA	NA
2	Create Ansible Repositories	Create ansible local, remote and virtual repositories	Create Ansible Repositories	Repositories
3	Manage Ansible Repositories	Manage ansible local, remote and virtual repositories	Manage Ansible Repositories	NA

##### 7.6.1.2 | Application Module - Ansible Packages

The activities of the application user are outlined in the following table:

# JFrog Artifactory Documentation

## Displayed in the header

#	Task	Description	For more information, see...
1	Artifactory Login	Log into your artifactory with the credentials.	NA
2	Set Me Up	Set up ansible client CLI to work with ansible repositories for publishing collections and installing collections and roles	<a href="#">View Set Me Up Ansible Code Snippets</a> <a href="#">Configure Ansible Client to Work with Artifactory</a>
3	Publish Ansible collections  Install Ansible collections and roles	Publish ansible collection to ansible repositories  Install ansible collections and roles from ansible repositories	<a href="#">Publish Ansible Collections</a> <a href="#">Install Ansible Collections and Roles</a>
4	Manage Ansible Collections and Roles	Manage ansible collections and roles	<a href="#">Manage Ansible Collections and Roles</a>

### 7.6.1.3 | Types of Ansible Repositories

There are 3 types of repositories, as follows:

- Local Repositories
- Remote Repositories
- Virtual Repositories

#### Ansible Local Repositories

Ansible Local Repositories are physical, locally managed repositories into which you can deploy artifacts. Using local repositories, Artifactory gives you a central location to store your internal binaries. Through repository replication, you can even share binaries with teams that are located in remote locations.

A local Ansible repository is where you deploy and host your internal Ansible collections. Ansible registry can host collections, which are your Ansible repositories. Once your collections 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.

#### Ansible Remote Repositories

Ansible Remote Repositories in Artifactory act as proxies for repositories on remote servers. Artifactory first checks its local cache for the requested package. If the package is not found in the cache, Artifactory retrieves it from the remote repository via the Internet. Once retrieved, the file is cached locally in Artifactory, making it available for any future requests without needing an internet connection. Importantly, only the requested package is cached, not the entire remote repository.

Artifactory supports proxying remote Ansible registries through remote repositories. A Remote repository in Artifactory serves as a caching proxy for a registry managed at a remote URL, the default is <https://galaxy.ansible.com>, but it can be any other registry or even an Ansible repository managed at a remote site by another instance of Artifactory.

Resources that are requested from a remote repository are cached on demand. You can remove downloaded resources from the remote repository cache, however, you can not manually push resources to a remote repository.

#### Ansible Virtual Repositories

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.

Artifactory supports virtual Ansible repositories. A Virtual repository aggregates collections from local and remote repositories that are included in the virtual repositories. This allows you to access collections hosted locally on local Ansible repositories, and remote collections and roles proxied by remote Ansible repositories, and access all of them from a single URL defined for the virtual repository.

Virtual repositories can be very useful since users will continue to work with them. Only the admin can manage the included repositories, and replace the default deployment target, while the changes will be transparent to the users.

### 7.6.2 | Administer Ansible Repositories

The following table outlines the activities you can perform on ansible repositories from administrator module:

Task	Description	For more information, see...
Create Ansible Repositories	Describes the steps to create ansible repositories	<a href="#">Create Ansible Repositories</a>
Manage Ansible Repositories	Describes the steps to manage ansible repositories	<a href="#">Manage Ansible Repositories</a>

#### 7.6.2.1 | Create Ansible Repositories

# JFrog Artifactory Documentation

## Displayed in the header

The following table outlines the creation and configuration of all ansible repositories:

Task	Description	For more information, see...
Create Local Ansible Repositories	Describes the steps to create and configure local ansible repositories.	<a href="#">Create Local Ansible Repositories</a>
Create Remote Ansible Repositories	Describes the steps to create and configure remote ansible repositories.	<a href="#">Create Remote Ansible Repositories</a>
Create Virtual Ansible Repositories	Describes the steps to create and configure virtual ansible repositories.	<a href="#">Create Virtual Ansible Repositories</a>

7.6.2.1.1 | [Create Local Ansible Repositories](#)

This section describes the steps to create and references to configure local ansible repositories.

To create a local Ansible repository, follow these steps:

1. Click **Administration** tab, and then click **Repositories**.

2. Click **Local** from **Create a Repository** drop-down list.

3. Click **Ansible** in the **Select Package Type** dialog.

4. Set **Repository Key** in the **New Local Repository** window.

5. For other **Basic** settings, refer to the [Basic Settings for Local Repositories](#).

6. For **Advanced** settings, refer to the [Advanced Settings for Local Repositories](#).

7. For **Replications** settings, refer to the Repository Replication.

8. Click **Create Local Repository**.

7.6.2.1.2 | [Create Remote Ansible Repositories](#)

This section describes the steps to create and references to configure remote ansible repositories.

To create a remote repository to proxy a remote Ansible registry, follow these steps:

1. Click **Administration** tab, and then click **Repositories**.

## JFrog Artifactory Documentation Displayed in the header

2. Click **Remote** from **Create a Repository** drop-down list.

3. Click **Ansible** in the **Select Package Type** dialog.

4. Set **Repository Key** in the **New Local Repository** window.

5. For other **Basic** settings, refer to the [Basic Settings for Remote Repositories](#).
6. For **Advanced** settings, refer to the [Advanced Settings for Remote Repositories](#).

7. For **Replications** settings, refer to the [Repository Replication](#).

8. Click **Create Remote Repository**.

7.6.2.1.3 | [Create Virtual Ansible Repositories](#)

This section describes the steps to create and references to configure virtual ansible repositories.

**To create a virtual Ansible repository, follow these steps:**

1. Click **Administration** tab, and then click **Repositories**.
2. Click **Virtual** from **Create a Repository** drop-down list.

## JFrog Artifactory Documentation Displayed in the header

3. Select **Ansible** from the **Select Package Type** dialog.

4. Set **Repository Key** in the **New Local Repository** window.

5. For other settings, refer to the [Basic Settings for Virtual Repositories](#) and [Advanced Settings for Virtual Repositories](#).

6. Click **Create Virtual Repository**.

### Tip

We recommend referencing a Virtual Repository URL as a repository. You can reconfigure and aggregate other external sources and local repositories of Ansible roles and collections you deploy.

# JFrog Artifactory Documentation

## Displayed in the header

### 7.6.2.2 | Manage Ansible Repositories

The managing activities of the local ansible repositories are outlined in the following table:

Task	Description	For more information, see...
Recalculate Index of Local Ansible Repositories	Describes the steps to recalculate the index of the local ansible repositories.	<a href="#">Recalculate Index of Local Ansible Repositories</a>
Edit Ansible Repositories	Describes the steps to edit the ansible repositories.	<a href="#">Edit Ansible Repositories</a>
Delete Ansible Repositories	Describes the steps to delete the ansible repositories.	<a href="#">Delete Ansible Repositories</a>

#### 7.6.2.2.1 | Recalculate Index of Local Ansible Repositories

The section describes how to recalculate the index of a local ansible repository.

**To recalculate a local Ansible repository index, follow these steps:**

1. Click **Administration** tab, and then click **Repositories**.
2. Click **Local** in the **Repositories** window.
3. Hover the mouse pointer over the repository you want to delete, click **more** icon, and then click **Recalculate Index**.

#### 7.6.2.2.2 | Edit Ansible Repositories

The following table outlines the editing of all ansible repositories:

Task	Description	For more information, see...
Edit Local Ansible Repositories	Describes the steps to edit the local ansible repositories.	<a href="#">Edit Local Ansible Repositories</a>
Edit Remote Ansible Repositories	Describes the steps to edit the remote ansible repositories.	<a href="#">Edit Remote Ansible Repositories</a>
Edit Virtual Ansible Repositories	Describes the steps to edit the virtual ansible repositories.	<a href="#">Edit Virtual Ansible Repositories</a>

# JFrog Artifactory Documentation

## Displayed in the header

7.6.2.2.1 | [Edit Local Ansible Repositories](#)

This section describes the steps to edit local ansible repositories.

**To edit a local Ansible repository, follow these steps:**

1. Click **Administration** tab, and then click **Repositories**.
2. Click **Local** in the **Repositories** window.

3. Click the Ansible Repository you want to edit from the list.

4. Edit the fields you want to make changes.

5. Click **Save**.

7.6.2.2.2.2 | [Edit Remote Ansible Repositories](#)

This section describes the steps to edit remote ansible repositories.

**To edit a remote Ansible repository, follow these steps:**

1. Click **Administration** tab, and then click **Repositories**.
2. Click **Remote** in the **Repositories** window.

3. Click the Ansible Repository you want to edit from the list.

4. Edit the fields you want to make changes.

5. Click **Save**.

7.6.2.2.3 | [Edit Virtual Ansible Repositories](#)

This section describes the steps to edit virtual ansible repositories.

**To edit a virtual Ansible repository, follow these steps:**

1. Click **Administration** tab, and then click **Repositories**.
2. Click **Virtual** in the **Repositories** window.

# JFrog Artifactory Documentation

## Displayed in the header

3. Click the Ansible Repository you want to edit from the list.

4. Edit the fields you want to make changes.

5. Click **Save**.

### 7.6.2.2.3 | [Delete Ansible Repositories](#)

The following table outlines the deletion of all ansible repositories:

Task	Description	For more information, see...
Delete Local Ansible Repositories	Describes the steps to delete the local ansible repositories.	<a href="#">Delete Local Ansible Repositories</a>
Delete Remote Ansible Repositories	Describes the steps to delete the remote ansible repositories.	<a href="#">Delete Remote Ansible Repositories</a>
Delete Virtual Ansible Repositories	Describes the steps to delete the virtual ansible repositories.	<a href="#">Delete Virtual Ansible Repositories</a>

#### 7.6.2.2.3.1 | [Delete Local Ansible Repositories](#)

This section describes the steps to delete local ansible repositories.

**To delete a local Ansible repository, follow these steps:**

1. Click **Administration** tab, and then click **Repositories**.
2. Click **Local** in the **Repositories** window.
3. Hover the mouse pointer over the repository you want to delete, click **more** icon, and then click **Delete**.

7.6.2.2.3.2 | [Delete Remote Ansible Repositories](#)

This section describes the steps to delete remote ansible repositories.

To delete a remote Ansible repository, follow these steps:

1. Click **Administration** tab, and then click **Repositories**.
2. Click **Remote** in the **Repositories** window.
3. Hover the mouse pointer over the repository you want to delete, and then click **Delete** icon.

7.6.2.2.3.3 | [Delete Virtual Ansible Repositories](#)

This section describes the steps to delete virtual ansible repositories.

### To delete a virtual Ansible repository, follow these steps:

1. Click **Administration** tab, and then click **Repositories**.
2. Click **Virtual** on the **Repositories** window.
3. Hover the mouse pointer over the repository you want to delete, and then click **Delete** icon.

7.6.3 | [Set Me Up Ansible](#)

The section describes working with Artifactory as a logged in user or anonymous user.

### Authentication

You can work with Artifactory as a logged in user or as an anonymous user.

#### Work with Artifactory using Login Credentials

Log in to your Artifactory with your credentials.

#### Work with Artifactory using Anonymous Access

By default, Artifactory allows anonymous access to Ansible repositories. This is defined in the Administration 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 users must enter their username and password as described in Setting Your Credentials above.

7.6.4 | [View Set Me Up Ansible Code Snippets](#)

The section describes the steps to view Ansible Set Me Up code snippets.

### To view **Configure**, **Publish** and **Install** code snippets, follow these steps:

1. Click **Application** tab, and then click **Artifacts** under **Artifactory**.
2. Locate the repository you want to configure, and then click **Set Me Up**.

**Note**

Alternatively, you can also choose the repository from the **Repository** drop-down list under **Set UP An Ansible Client** modal.

3. Enter your JFrog account password to generate a token, and then click **Generate Token & Create Instructions**.

To learn more about **Set Me Up** code snippets, refer to the following sections:

# JFrog Artifactory Documentation

## Displayed in the header

Task	Description	For more information, see...
Configure Ansible Client to Work with Artifactory	Configures ansible client to work with artifactory	<a href="#">Configure Ansible Client to Work with Artifactory</a>
Publish Ansible Collections	Publishes ansible collections	<a href="#">Publish Ansible Collections</a>
Install Ansible Collections and Roles	Installs ansible collections and roles	<a href="#">Install Ansible Collections and Roles</a>

### 7.6.4.1 | Configure Ansible Client to Work with Artifactory

Once you have created your Ansible repository, you can get helpful code snippets from the UI to resolve collections and roles using the Ansible client.

You can publish ansible collections, and install ansible collections and roles using any one of the following methods:

- **Configuration Files**

- For collections, Server in ansible.cfg file
- For roles, Server in ansible.cfg and .netrc files

- **Server Explicit**

Pass server and token details from the CLI.

#### Configure Ansible Client

To configure ansible client to work with collections and roles using configuration files, refer to the following sections:

##### For Collections

Task	Description	For more information, see...
Configure Ansible Client to Work with Collections using ansible.cfg File	Describes the steps to configure ansible client to work with collections using ansible.cfg file	<a href="#">Configure Ansible Client to Work with Collections using ansible.cfg File</a>

##### For Roles

Task	Description	For more information, see...
Configure Ansible Client to Work with Roles using ansible.cfg and .netrc Files	Describes the steps to configure ansible client to work with roles using ansible.cfg and .netrc files	<a href="#">Configure Ansible Client to Work with Roles using .netrc File</a>

##### Server Explicit

To publish collections, and install collections and roles server-explicitly (without configuration files), refer to the following sections:

##### For Collections

Task	Description	For more information, see...
Publish Ansible Collections via Ansible CLI without ansible.cfg File	Describes the steps to publish ansible collections via ansible CLI without ansible.cfg file	<a href="#">Publish Ansible Collections via Ansible CLI without ansible.cfg File - Server Explicit</a>
Install Ansible Collections without ansible.cfg File	Describes the steps to install ansible collections via ansible CLI without ansible.cfg file	<a href="#">Install Ansible Collections - Server Explicit</a>

##### For Roles

Task	Description	For more information, see...
Install Ansible Roles without ansible.cfg and .netrc Files	Describes the steps to install ansible roles via ansible CLI without ansible.cfg and .netrc files	<a href="#">Install Ansible Roles without .netrc - Server Explicit</a>

### 7.6.4.1.1 | Configure Ansible Client to Work with Collections using ansible.cfg File

# JFrog Artifactory Documentation

## Displayed in the header

This section describes the steps to configure Ansible client to work with collections using `ansible.cfg` file.

To locate configure code snippet based on your repository selection, follow these steps:

1. Click **Application** tab, and then click **Artifacts** under **Artifactory**.
2. Locate the repository you want to configure, and then click **Set Me Up**.

### Note

Alternatively, you can also choose the repository from the **Repository** drop-down list under **Set UP An Ansible Client** modal.

3. Enter your JFrog account password to generate a token, and then click **Generate Token & Create Instructions**.

# JFrog Artifactory Documentation Displayed in the header

4. Click the **Configure** tab, copy and paste the snippet of **Collections** to your ansible.cfg file.

The sample ansible.cfg snippet structure is as follows:

## Note

Make sure to replace the placeholders in bold with your JFrog host domain, desired repository path and token.

```
[galaxy]
server_list = <REPOSITORY_NAME>

[galaxy_server.<REPOSITORY_NAME>]
url=https://<SERVER_NAME>/artifactory/api/ansible/<REPOSITORY_NAME>
token=<TOKEN>
```

- <REPOSITORY\_NAME>: Name of the repository/repository key.
- <SERVER\_NAME>: URL of your Jfrog instance (AKA JPD).
- <TOKEN>: Your JPD access token

For example:

```
[galaxy]
server_list = ansible-virtual

[galaxy_server.ansible-virtual]
url=https://my-awesome.jfrog.io/artifactory/api/ansible/ansible-virtual
token=<TOKEN>
```

## Note

The server\_list parameter can include multiple repositories separated by a comma.

For example: server\_list = ansible-virtual, ansible-local

Alternatively, you can choose to use the Artifactory <URL> and <Token> explicitly in your Publish and Install commands.

7.6.4.1.2 | Configure Ansible Client to Work with Roles using .netrc File

Artifactory supports the installation of roles resolved via remote repository.

For resolving roles via Artifactory, configure your Ansible registry credentials in your .netrc file.

To locate configure code snippet based on your repository selection, follow these steps:

1. Click **Application** tab, and then click **Artifacts** under **Artifactory**.
2. Locate the repository you want to configure, and then click **Set Me Up**.

**Note**

Alternatively, you can also choose the repository from the **Repository** drop-down list under **Set UP An Ansible Client** modal.

3. Enter your JFrog account password to generate a token, and then click **Generate Token & Create Instructions**.

4. Click the **Configure** tab, copy and paste the snippet of **Roles** to your `.netrc` file.

## Note

To configure roles, you must add your credentials to the `.netrc` file in addition to `ansible.cfg` file. To learn how to configure `ansible.cfg`, refer to [Configure Ansible Client to Work with Collections using `ansible.cfg` File](#)

Grant `chmod 600 .netrc` privileges.

The sample `.netrc` snippet structure is as follows:

## Note

Make sure to replace the placeholders in bold with your JFrog host domain and credentials.

```
machine <SERVER_NAME> login <YOUR_JFROG_USERNAME> password/identity-token <YOUR_JFROG_PASSWORD>/<YOUR_JFROG_IDENTITY-TOKEN>
```

- <SERVER\_NAME>: URL of your JPD.
- <YOUR\_JFROG\_USERNAME>: Your JPD username.
- <YOUR\_JFROG\_PASSWORD>/<YOUR\_JFROG\_IDENTITY\_TOKEN>: Your JPD password / Identity Token

For example:

```
machine my-awesome.jfrog.io login admin password cmVmdGtu0jAx0jE3NTA0jfauuuYYt3u73totu1JqRVNSMWhUS1htQX1zb21TcnVH
```

### 7.6.4.2 | Publish Ansible Collections

The following table outlines the different approaches for publishing ansible collection:

Task	Description	For more information, see...
<b>Via CLI Server in <code>ansible.cfg</code> file</b> Publish Ansible Collections via Ansible CLI using <code>ansible.cfg</code> File	Describes the steps to publish ansible collections via ansible CLI using <code>ansible.cfg</code> file	<a href="#">Publish Ansible Collections via Ansible CLI - Server in <code>ansible.cfg</code> File</a>
<b>Via CLI Server Explicit</b> Publish Ansible Collections via Ansible CLI without <code>ansible.cfg</code> File	Describes the steps to publish ansible collections via ansible CLI without <code>ansible.cfg</code> file	<a href="#">Publish Ansible Collections via Ansible CLI without <code>ansible.cfg</code> File - Server Explicit</a>
<b>Via UI and API</b> Deploy Ansible Collections via UI and API	Describes the steps to deploy ansible collections via UI and UPI	<a href="#">Deploy Ansible Collections via UI and API.</a>

7.6.4.2.1 | [Publish Ansible Collections via Ansible CLI - Server in `ansible.cfg` File](#)

# JFrog Artifactory Documentation

## Displayed in the header

This section describes the steps to publish Ansible collection with `ansible.cfg` file via Ansible client CLI.

To locate publish collections code snippet based on your repository selection, follow these steps:

1. Click **Application** tab, and then click **Artifacts** under **Artifactory**.
2. Locate the repository you want to configure, and then click **Set Me Up**.

### Note

Alternatively, you can also choose the repository from the **Repository** drop-down list under **Set UP An Ansible Client** modal.

3. Enter your JFrog account password to generate a token, and then click **Generate Token & Create Instructions**.

# JFrog Artifactory Documentation

## Displayed in the header

4. Click the **Publish** tab, copy and paste the snippet of **Server in ansible.cfg file** to your ansible client.

### Note

You must configure `ansible.cfg` file in your workspace to use **Server in ansible.cfg file** approach. To learn how to configure, refer to the [Configure Ansible Client to Work with Collections using ansible.cfg File](#) section.

5. Replace the placeholder `<NAMESPACE.COLLECTION_NAME>` as appropriate, and then run the command.

The sample snippet structure is as follows:

### Note

Make sure to replace the placeholder in bold with the path to your archive.

```
ansible-galaxy collection publish <NAMESPACE.COLLECTION_NAME>
```

For example:

```
ansible-galaxy collection publish geerlingguy.k8s.tar.gz
```

7.6.4.2.2 | Publish Ansible Collections via Ansible CLI without ansible.cfg File - Server Explicit

This section describes the steps to publish collections via Ansible client CLI, without `ansible.cfg` File.

To locate publish code snippet based on your repository selection, follow these steps:

1. Click **Application** tab, and then click **Artifacts** under **Artifactory**.
2. Locate the repository you want to configure, and then click **Set Me Up**.

**Note**

Alternatively, you can also choose the repository from the **Repository** drop-down list under **Set UP An Ansible Client** modal.

3. Enter your JFrog account password to generate a token, and then click **Generate Token & Create Instructions**.

4. Click the **Publish** tab, copy and paste the snippet of **Server explicit** to your ansible client.

5. Replace the placeholder <NAMESPACE.COLLECTION\_NAME> as appropriate, and then run the command.

The sample snippet structure is as follows:

**Note**

Make sure to replace the placeholders in bold with the namespace and collection name, server name, repository name, and token.

```
ansible-galaxy collection publish <NAMESPACE.COLLECTION_NAME> -s https://<SERVER_NAME>/artifactory/api/ansible/<REPOSITORY_NAME> --token=<TOKEN>
```

**For example:**

```
ansible-galaxy collection publish gearlingguy.k8s.tar.gz -s https://my-awesome.jfrog.io/artifactory/api/ansible/ansible-virtual --token=<TOKEN>
```

7.6.4.2.3 | Deploy Ansible Collections via UI and API

This section describes the steps to deploy ansible collections via UI and API.

Once you have configured your local machine to install collections from your Ansible local repository, you may also deploy Ansible collections to the same repository using the UI or the REST API.

Through the REST API, you also have the option to deploy by checksum or by deploying from an archive.

For example, to deploy an Ansible collection into a repository called ansible-local, you could use the following:

```
curl -u<USERNAME>:<PASSWORD> -XPUT http://localhost:8080/artifactory/ansible-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.

# JFrog Artifactory Documentation

## Displayed in the header

### 7.6.4.3 | Install Ansible Collections and Roles

The following table outlines installing ansible collections and roles:

Task	Description	For more information, see...
Install Ansible Collections	Describes the steps to install ansible collections with different approaches	<a href="#">Install Ansible Collections</a>
Install Ansible Roles	Describes the steps to install ansible roles with different approaches	<a href="#">Install Ansible Roles</a>

#### 7.6.4.3.1 | [Install Ansible Collections](#)

The following table outlines installing ansible collections using different approaches:

Task	Description	For more information, see...
<b>Server in ansible.cfg file</b> <a href="#">Install Ansible Collections using ansible.cfg File</a>	Describes the steps to install ansible collections via ansible CLI using ansible.cfg file	<a href="#">Install Ansible Collections using ansible.cfg File</a>
<b>Server explicit</b> <a href="#">Install Ansible Collections without ansible.cfg File</a>	Describes the steps to install ansible collections via ansible CLI without ansible.cfg file	<a href="#">Install Ansible Collections - Server Explicit</a>

#### 7.6.4.3.1.1 | [Install Ansible Collections using ansible.cfg File](#)

This section describes the steps to install Ansible collection using ansible.cfg file via Ansible client CLI.

To locate install collections code snippet based on your repository selection, follow these steps:

1. Click **Application** tab, and then click **Artifacts** under **Artifactory**.
2. Locate the repository you want to configure, and then click **Set Me Up**.

### Note

Alternatively, you can also choose the repository from the **Repository** drop-down list under **Set UP An Ansible Client** modal.

3. Enter your JFrog account password to generate a token, and then click **Generate Token & Create Instructions**.

4. Click the **Install** tab, copy and paste the **Server in ansible.cfg file** snippet of **Collections** to your ansible client.

**Note**

You must configure ansible.cfg file in your workspace to use **Server in ansible.cfg file** approach. To learn how to configure, refer to the [Configure Ansible Client to Work with Collections using ansible.cfg File](#) section.

5. Replace the placeholder <NAMESPACE.COLLECTION\_NAME> as appropriate, and then run the command.

The sample snippet structure is as follows:

# JFrog Artifactory Documentation

## Displayed in the header

### Note

Make sure to replace the placeholders in bold with the namespace and collection name.

```
ansible-galaxy collection install <NAMESPACE.COLLECTION_NAME>.tar.gz
```

For example:

```
ansible-galaxy collection install geerlingguy.k8s.tar.gz
```

7.6.4.3.1.2 | Install Ansible Collections - Server Explicit

This section describes the steps to install Ansible collections without `ansibl.cfg` file via Ansible client CLI.

To locate install collections code snippet based on your repository selection, follow these steps:

1. Click **Application** tab, and then click **Artifacts** under **Artifactory**.
2. Locate the repository you want to configure, and then click **Set Me Up**.

### Note

Alternatively, you can also choose the repository from the **Repository** drop-down list under **Set UP An Ansible Client** modal.

3. Enter your JFrog account password to generate a token, and then click **Generate Token & Create Instructions**.

4. Click the **Install** tab, copy and paste the **Server explicit** snippet of **Collections** to your ansible client.

5. Replace the placeholder <NAMESPACE.COLLECTION\_NAME> as appropriate, and then run the command.

The sample snippet structure is as follows:

**Note**

Make sure to replace the placeholders in bold with the namespace and collection name, server name, repository name, and token.

```
ansible-galaxy collection install <NAMESPACE.COLLECTION_NAME>.tar.gz -s https://<SERVER_NAME>/artifactory/api/ansible/<REPOSITORY_NAME> --token=<TOKEN>
```

For example:

# JFrog Artifactory Documentation

## Displayed in the header

```
ansible-galaxy collection install gearlingguy.k8s.tar.gz -s https://my-awesome.jfrog.io/artifactory/api/ansible/ansible-virtual --token=<TOKEN>
```

### 7.6.4.3.2 | Install Ansible Roles

The following table outlines installing ansible roles using different approaches:

Task	Description	For more information, see...
<b>Server in ansible.cfg file</b> Install Ansible Roles using ansible.cfg and .netrc Files	Describes the steps to install ansible roles via ansible CLI using ansible.cfg and .netrc files	<a href="#">Install Ansible Roles using .netrc File</a>
<b>Server explicit</b> Install Ansible Roles without ansible.cfg and .netrc Files	Describes the steps to install ansible roles via ansible CLI without ansible.cfg and .netrc files	<a href="#">Install Ansible Roles without .netrc - Server Explicit</a>

### 7.6.4.3.2.1 | Install Ansible Roles using .netrc File

This section describes the steps to install Ansible roles using .netrc file via Ansible client CLI.

**To locate install roles code snippet based on your repository selection, follow these steps:**

1. Click **Application** tab, and then click **Artifacts** under **Artifactory**.
2. Locate the repository you want to configure, and then click **Set Me Up**.

### Note

Alternatively, you can also choose the repository from the **Repository** drop-down list under **Set UP An Ansible Client** modal.

3. Enter your JFrog account password to generate a token, and then click **Generate Token & Create Instructions**.

4. Click the **Install** tab, copy and paste the **Server in ansible.cfg file** snippet of **Roles** to your ansible client.

**Note**

You must configure `.netrc` file in your workspace to use **Server in ansible.cfg file** approach. To learn how to configure, refer to the [Configure Ansible Client to Work with Roles using .netrc File](#) section.

5. Replace the placeholder `<NAMESPACE.ROLE_NAME>` as appropriate, and then run the command.

The sample snippet structure is as follows:

### Note

Make sure to replace the placeholders in bold with the namespace and role name.

```
ansible-galaxy role install <NAMESPACE.ROLE_NAME>.tar.gz
```

For example:

```
ansible-galaxy role install gearlingguy.k8s.tar.gz
```

7.6.4.3.2.2 | Install Ansible Roles without .netrc - Server Explicit

This section describes the steps to install Ansible roles without .netrc file via Ansible client CLI.

To locate install roles code snippet based on your repository selection, follow these steps:

1. Click **Application** tab, and then click **Artifacts** under **Artifactory**.
2. Locate the repository you want to configure, and then click **Set Me Up**.

### Note

Alternatively, you can also choose the repository from the **Repository** drop-down list under **Set UP An Ansible Client** modal.

3. Enter your JFrog account password to generate a token, and then click **Generate Token & Create Instructions**.

4. Click the **Install** tab, copy and paste the **Server explicit** snippet of **Roles** to your ansible client.

5. Replace the placeholder <NAMESPACE.ROLE\_NAME> as appropriate, and then run the command.

The sample snippet structure is as follows:

**Note**

Make sure to replace the placeholders in bold with the namespace and role name, server name, repository name, and token.

```
ansible-galaxy role install <NAMESPACE.ROLE_NAME>.tar.gz -s https://<SERVER_NAME>/artifactory/api/ansible/<REPOSITORY_NAME> --token=<TOKEN>
```

For example:

# JFrog Artifactory Documentation

## Displayed in the header

```
ansible-galaxy role install gearlingguy.k8s.tar.gz -s https://my-awesome.jfrog.io/artifactory/api/ansible/ansible-virtual --token=<TOKEN>
```

### 7.6.5 | Manage Ansible Collections and Roles

This section describes how to manage ansible collections and roles.

Task	Description	For more information, see...
Search Ansible Collections	Describes the steps to search ansible collections	<a href="#">Search Ansible Collections</a>
List Ansible Packages Versions/Tags	Describes the steps to list ansible packages versions/tags	<a href="#">List Ansible Packages Versions/Tags</a>
View Ansible BuildInfo	Describes the steps to view ansible buildinfo	<a href="#">View Ansible BuildInfo</a>
View Individual Ansible Collections Information	Describes the steps to view individual ansible collections information	<a href="#">View Individual Ansible Collections Information</a>
Clean Up Ansible Local Artifactory Cache	Describes the steps to clean up ansible local artifactory cache	<a href="#">Clean Up the Ansible Local Artifactory Cache</a>

#### 7.6.5.1 | Search Ansible Collections

You can search for Ansible collections by name using the [Artifact Package Search](#) or through the REST API.

[Search Ansible Collections via UI](#)

Artifactory supports a variety of ways to search for artifacts. For details, please refer to [Browse and Search Artifacts](#). However, the collection may not be available immediately after being published, for the following reasons:

- When publishing collections to a local repository, Artifactory calculates the search index asynchronously and will wait to index the newly published collections.
- Since a virtual repository may contain local repositories, a newly published collection may not be available immediately for the same reason.
- In the case of remote repositories, a new collection will only be found once Artifactory checks for it according to the Retrieval Cache Period setting.

[Search Ansible Collections via API](#)

Search Artifactory using REST API. Refer to [Artifact Search API](#).

#### Tip

Artifactory annotates each deployed or cached Ansible collection with the following properties:

```
ansible.name, ansible.namespace, ansible.fullname, ansible.version, ansible.tags, ansible.requires and ansible.id
```

You can use Property Search to search for Ansible collections according to their name or version.

#### 7.6.5.2 | List Ansible Packages Versions/Tags

The list displays information about the package versions.

To learn more about list versions, refer to the [Viewing Package Information](#).

## JFrog Artifactory Documentation Displayed in the header

### 7.6.5.3 | View Ansible BuildInfo

Build-info is all the information collected by the build agent, which includes details about the build. The build-info includes a list of project modules, artifacts, dependencies, environment variables and more. When using one of the JFrog clients to build the code, the client can collect the build-info and publish it to Artifactory. When the build-info is published to Artifactory, all the published details become visible in the Artifactory UI.

#### View Ansible BuildInfo via UI

Refer to the View Build Number Information documentation.

#### View Ansible BuildInfo via API

Refer to the Build Info REST API documentation.

### 7.6.5.4 | View Individual Ansible Collections Information

Artifactory lets you view selected Ansible collections metadata and their contents directly from the UI.

#### To view Ansible info, follow these steps:

1. Click **Application** tab, and then click **Artifacts** under **Artifactory**.
2. Drill down in the **Tree Browser**, select the **tar.gz** file you want to inspect, and view the **Package Info**, **Dependencies** and **Contents** from the **Ansible Info** tab.

### 7.6.5.5 | Clean Up the Ansible Local Artifactory Cache

The Ansible client saves caches of downloaded roles and collections and their JSON metadata responses (called `.cache.json`).

The JSON metadata cache files contain the Ansible roles and collections metadata.

We recommend removing the Ansible caches, both roles and collections 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://galaxy.ansible.com/>

## 7.7 | Bower Repositories

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 <https://registry.bower.io>) 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.

### 7.7.1 | Set Up a Bower Repository

You can set up the following repository types:

- Local repositories
- Remote repositories
- Virtual repositories

Follow the steps according to each repository type below.

#### 7.7.1.1 | Local Bower Repositories

To enable calculation of Bower package metadata, in the **Administration** module, go to **Repositories| Repositories | Local** and click on **New Local Repository**. Select **Bower** from the **Select Package Type** dialog to create your local Bower repository.

#### Integration Benefits

JFrog Artifactory and Bower Registries

##### 7.7.1.1.1 | Deploy 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.

#### 7.7.1.2 | Remote Bower 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 Repositories** which serves as a caching proxy for [github.com](https://github.com). If necessary, you can do the same for [bitbucket.org](https://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](https://github.com) as well as the public Bower registry follow the steps below:

1. Create a new remote repository in the **Administration** module, under **Repositories | Repositories | Remote**, click "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.

3. In the **Bower Settings** section, select **GitHub** as the **Git Provider**.

Finally, click "Save & Finish"

### 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>.

#### 7.7.1.3 | Virtual Bower 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.

#### 7.7.1.4 | Bower Repositories Advanced Configuration

The fields under **External Dependency Rewrite** are connected to automatically rewriting external dependencies for Bower packages that need them.

Field	Description
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 Allow List	An Allow List of Ant-style path expressions that specify where external dependencies may be downloaded from. By default, this is set to ** which means that dependencies may be downloaded from any external source.  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 ** expression).

#### 7.7.2 | Use 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 Cloud, the URL would be:

`https://<server name>.jfrog.io/artifactory/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.

#### 7.7.2.1 | Use CLI with 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

#### Tip

We recommend referencing a Virtual Repositories 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
```

### 7.7.2.2 | Use CLI with 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):

```
{  
  "registry": "http://localhost:8081/artifactory/api/bower/bower-repo"  
}
```

#### .bowerrc file location

Windows: %userprofile%\.bowerrc

Linux: ~/.bowerrc

#### Tip

We recommend referencing a Virtual Repositories 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
```

### 7.7.3 | Work with Bower 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:

# JFrog Artifactory Documentation

## Displayed in the header

### 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.

#### 7.7.4 | Clean 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 <https://registry.bower.io>.

To clear the bower cache use:

##### Clean Bower Cache

```
bower cache clean
```

#### 7.7.5 | Automatically Rewrite 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 downloading a 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.

### 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](https://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}}"
```

# JFrog Artifactory Documentation

## Displayed in the header

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$$$art-shorthand-resolver$$<username>-  
mypackagetest-master.tar.gz  
bower mypackagetest$$$art-shorthand-resolver$$<username>-mypackagetest-master.tar.gz          resolveart://<username>/mypackagetest$$$art-shorthand-resolver$$<username>-  
mypackagetest-master.tar.gz  
bower mypackagetest$$$art-shorthand-resolver$$<username>-mypackagetest-master.tar.gz          resolvedart://<username>/mypackagetest$$$art-shorthand-resolver$$<username>-  
mypackagetest-master.tar.gz
```

### 7.7.6 | Register 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
```

### 7.7.7 | View 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.

### 7.8 | Cargo Package Registry

From JFrog Artifactory 7.17.4, the Cargo registry is supported for the Rust programming language, giving you full control of your deployment and resolving Cargo packages. Cargo downloads your Rust package's dependencies, compiles your packages, makes distributable packages, and uploads them to crates.io, the Rust community's package registry. You can contribute to this book on [GitHub](#).

### Important

Cargo repositories require enabling the Custom Base URL for the Artifactory instance. For more information, see General Settings.

### Note

We support Cargo Sparse indexing from Artifactory 7.46.3 and by default from 7.58.0. If you still use Git indexing, you might experience the following issues and limitations:

- Repository initialization might experience stability issues.
- The Git implementation does not support proxy in remote repositories.
- Performance and build time might be slightly slower.

Therefore, we strongly recommend updating to Sparse indexing. For more information, see [Index Cargo Repositories Using Sparse Indexing](#)

### About Rust Programming Language

Rust is a programming language designed for performance and safety, with an emphasis on safety concurrency. A crate is a compilation unit in Rust. Using Cargo, you can publish libraries on crates.io, organize large projects with a workspace, install binaries from crates.io and extend Cargo using custom commands.

### Learn More

[Cheatsheet for managing applications using Rust & Cargo](#)

Cargo repositories in Artifactory offer the following benefits:

- Secure and private local Cargo repositories with fine-grained access control
- The ability to proxy remote Cargo resources and cache downloaded Cargo packages to keep you independent of the network and the remote resource
- Metadata calculation of the Cargo packages hosted in the Artifactory local repositories
- Version management: Archiving older versions of the packages uploaded to local repositories
- Source and binaries management

### Supported Cargo Version

Artifactory supports Cargo version 1.49.0 and above.

#### 7.8.1 | Set Up a Cargo Repository

You can set up the following repository types:

- Local Cargo Repositories
- Remote Cargo Repositories

Virtual Cargo Repositories are not supported.

##### 7.8.1.1 | Local Cargo Repositories

To enable the calculation of Cargo metadata, in the Administration module, go to [Repositories| Repositories | Local](#) and select **Cargo** as the **Package Type** when you create your local repository.



### Prerequisite

Prior to setting a local repository, you will need to configure a Custom Base URL for the Artifactory instance. For more information, see General Settings.

#### Local Cargo Repository Layout

You will need to maintain a specific path structure to manage the Cargo packages that are uploaded to Cargo local repositories.

Cargo Source packages are automatically uploaded by default to the relative path: `crates/{package_name}/{package_name}-{version}.crate`.

#### Yank and Un-yank Cargo Crates in Local Repositories

Artifactory supports yanking and un-yanking crates in local repositories.

##### Cargo Yank/ Un-yank

```
cargo yank hello_world --vers 0.1.4 --token "Bearer (token)"  
cargo yank hello_world --vers 0.1.4 --token "Bearer (token)" --undo
```

### Yank/ Un-yank Smart Repositories

To synchronize yanking in Smart remote repositories/replications, the properties must be synced.

#### 7.8.1.2 | Remote Cargo Repositories

##### Prerequisite

Prior to setting a remote repository, you will need to configure a Custom Base URL for the Artifactory instance that is required to support. For more information, see General Settings.

You can create remote Cargo repositories to proxy and cache remote repositories or other Artifactory instances.

The **Registry URL** has been added to Cargo remote repositories, to reflect the index (git) location:

- For external repositories, the registry URL should be the same as the repository URL. For example: <https://github.com/rust-lang/crates.io-index>.
- For Smart remote repositories, the URL should be the same as the repository URL, for example: [https://RTURL/artifactory/\\$repo](https://RTURL/artifactory/$repo), while the registry URL should point to the index path, for example: <https://RTURL/artifactory/api/cargo/cargoh/index/>.

#### 7.8.1.3 | Community Assistance To Support Cargo Virtual Repositories

Currently, JFrog Artifactory cannot support Cargo Virtual Repositories due to a Cargo Client limitation that does not provide a method to override dependencies for all Cargo Artifactory users.

##### What is the problem?

Cargo packages have a configuration file embedded into the package which lists the package's dependencies. Each dependency contains a URL field that is supposed to point to the same location of the package but more often the URLs point to a source in a different location than the package itself.

Specifically in the Jfrog Artifactory context the ability to modify the source URLs in a scalable manner is crucial for us to provide the Virtual Repository capability since it each dependency can point once to a Local repository and another to a Remote repository.

With this being said, Cargo have provided developers with a solution to override dependencies as documented in [Overriding Dependencies - The Cargo Book](#). Still, unfortunately, it is mostly for development use cases where a single user would set up their specific environment to point to local resources, and it does not provide a scalable method to override dependencies for all users consuming Cargo packages from Artifactory.

For further details please see the full discussions here:

- Add the ability to replace any source in `.cargo/config.toml` file
- Possibility to replace any source in `config.toml`

##### How can you help?

If you are a Cargo developer waiting for virtual repository support in Jfrog Artifactory, we would extremely appreciate your assistance in steering and providing a scalable solution.

Please reach out to us via email here: [artifactory-cargo-support@jfrog.com](mailto:artifactory-cargo-support@jfrog.com)

#### 7.8.2 | Resolve Cargo Packages

To resolve Cargo packages:

1. In the Application module, navigate to **Artifactory | Artifacts**.

# JFrog Artifactory Documentation

## Displayed in the header

2. In the Artifact Tree Browser, select a Cargo repository and click **Set Me Up**.

### Password Authentication Requirement

The 'Type Password' box is displayed only if you need to perform re-authentication opposite the JFrog Platform. This option will not be available in the UI for users applying external authentication methods such as SAML or OAuth.

To set your credentials for API calls such as publish and yank, add the following section to the credentials file under your Cargo home directory (for example `~/.cargo/credentials`):

```
[registries.artifactory]
token = "Bearer <TOKEN>"
```

3. Copy the Cargo code snippets to publish a Cargo package or configure your Rust client to resolve the artifacts in the selected repository.

#### 7.8.2.1 | Resolve Cargo Packages Using the Rust Command Line

1. In the Application module, navigate to **Artifactory | Artifacts**.
2. In the Artifact Tree Browser, select a Cargo repository and click **Set Me Up**.
3. Install a package using the Cargo build or install commands.

##### Cargo install

```
cargo install crate
```

##### Resolving Multiple Cargo Registries

To resolve multiple registries, add this optional flag `--registry (repositoryId)`.

#### Authentication: Allow Anonymous Downloads

The Cargo client does not send **any** authentication header when running install and search commands.

Select "Allow anonymous download and search" to block anonymous requests but still allow anonymous Cargo client downloads and performing search, to grant anonymous access specifically to those endpoints for the specific repository.

### 7.8.3 | Deploy Cargo Packages

You can deploy packages to a local Cargo repository using the Cargo Client, using the **Deploy** feature in the UI, or using a cURL request.

#### 7.8.3.1 | Deploy a Package Using the Cargo Client (Recommended)

To deploy a package, run the following Cargo publish command.

##### Cargo install

```
cargo publish or cargo publish --registry (registry id)
```

To override the credentials for that repository, run the following command.

##### Cargo install

```
cargo publish --token "Basic (base64 of user:password)" or cargo publish --token "Bearer (access token)"
```

#### 7.8.3.2 | Deploy a Cargo Package Using the UI

You can either drag and drop, or select a Cargo package to upload in **Deploy** in the UI. Artifactory will automatically identify if the package is a source or binary package.

##### Target Path

It is important to be aligned with the following layout to support this feature.

```
crates/{package_name}/{package_name}-{version}.crate
```

#### 7.8.3.3 | Deploy a Cargo Package Using cURL

To deploy a package using a cURL request.

##### Deploying Cargo Crates

```
curl -uadmin:password -XPUT "http://localhost:8082/artifactory/cargo-local/crates/package-1.0.0.crate" -T package-1.0.0.crate
```

When deploying directly (PUT request to a specific path), make sure the target path is a valid Cargo path.

```
crates/{package_name}/{package_name}-{version}.crate
```

Note that deploying a package to a different path will not identify the package as Cargo package, and will not invoke metadata indexing.

Also, as Artifactory relies on the information in the cargo.toml file, make sure that it contains the dependencies' registry URLs, if applicable. If you are not sure, it is always recommended to use the cargo publish command.

#### 7.8.4 | View Individual Cargo Package Information

Artifactory lets you view selected metadata of a Cargo package directly from the UI.

In the [Artifact Repository Browser](#), select your local Cargo repository and scroll down to find and select the package you want to inspect.

The metadata is displayed in the [Cargo Info](#) tab, or view in the [Packages](#) view.

#### 7.8.5 | Index Cargo Repositories

There are several ways to index Cargo repositories:

- [Using Sparse Indexing](#)
- [Using Git Indexing](#)
- [Reindex Cargo Repositories](#)

##### 7.8.5.1 | Index Cargo Repositories Using Sparse Indexing

From Artifactory version 7.46.3, Cargo sparse indexing is supported. Sparse Indexing allows you to control the connections that are entered into the index database. Sparse indexing also allows for:

- Faster build times, since no Git clones/ pulls are involved
- Using a proxy for Remote Cargo Repositories.

##### Note

Cargo sparse indexing is supported from Cargo version 1.68 and higher (from Cargo 1.60 it is supported as a nightly feature).

# JFrog Artifactory Documentation

## Displayed in the header

To enable Cargo sparse indexing:

- Via REST API: Add the "cargoInternalIndex": true flag to the Repository Configuration JSON file.
- Via the JFrog Platform UI: Select the **Enable sparse index support** checkbox in the General Settings section of the Cargo Repository Basic tab.

For existing repositories, reindex after enabling sparse indexing. For new repositories, no initialization time is required, as no Git clone is created behind the scenes.

### Rules and Guidelines

- When configuring as a Smart Remote repository, the URL should use the following syntax: `http://[domain]/artifactory/cargo-local`
- While the registry URL should use this syntax: `http://domain/artifactory/api/cargo/cargo-local/index`

To use the official Rust Remote Repository URL, create a remote repository with the URL and registry URL `https://index.crates.io`.

For more information, see the [Rust documentation](#).

#### 7.8.5.2 | Index Cargo Repositories Using Git Indexing

##### Note

From Artifactory version 7.58.x, the default method for Cargo indexing is sparse indexing. However, If you are using an Artifactory earlier than 7.46.3, or a Cargo client earlier than 1.60, you may need to use Cargo Git Indexing.

To configure the Cargo client to work opposite JFrog Artifactory using Git indexing:

1. Edit the configuration file under your Cargo home directory (for example `~/.cargo/config.toml`). In this example we use `artprod.mycompany` to represent the Artifactory:

```
# Makes artifactory the default registry and saves passing --registry parameter
[registry]
default = "artifactory"

[registries.artifactory]
index = "https://productdemo.jfrog.io/artifactory/git/bank32-cargo-local-2.git"

# For sending credentials in git requests.
# Not required if anonymous access is enabled
[net]
git-fetch-with-cli = true
```

2. To set your credentials for API calls such as publish and yank, add the following section to the credentials file under your Cargo home directory (for example `~/.cargo/credentials.toml`):

```
[registries.artifactory]
token = "Bearer pm-admin:<PASSWORD> (converted to base 64)"
```

### Git Support for Cargo Repositories

As the Cargo Client requires the Cargo registry to be a Git repository, the following Git support has been applied in Artifactory:

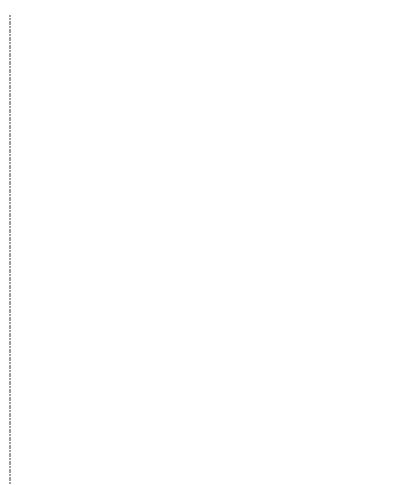
- An internal .git folder exists for each Cargo repository to reflect the index .git files. This folder is recreated after every reindexing process.
- An internal git directory in the Data directory of Artifactory has been added for each Cargo repository. This is a local clone that will be recreated following each restart or repository init.

#### 7.8.5.3 | Re-index a Cargo Repository

You can trigger an asynchronous re-indexing of a local Cargo repository either through the UI or using the REST API.

This will also reindex the git index and, as a result, will also index the remote repositories.

In the **Artifact Tree Browser**, select your Cargo repository, right-click and select **Recalculate Index** from the list. Requires Admin privileges.



To reindex a Cargo repository through the REST API, refer to [Calculate Cargo Repository Metadata](#).

Integration Benefits Cargo Registry

### 7.8.6 | Cargo Package Federation Limitation and Workaround

#### Limitation

Unlike other package managers, Cargo packages contain the URL of the registry where they have been uploaded to. The Cargo client does not support a global URL replacement method, so the client can't modify dynamic URLs. Artifactory cannot change the URLs in each Cargo package because this would change the package checksum and the package would be considered compromised by the client.

This limitation makes it difficult for Artifactory to enable Virtual repositories and Federation unless the following workaround solution is performed.

#### Cargo Rewrite Workaround Solution

Cargo recommends using Source Replacement on each developer machine to cope with the URL validation from different sources.

For each instance in the federation, configure `.cargo/config.toml` to identify all the other federation members. Use the `replace-with` option for each federation source with the target URL. To use FED1, add a source for FED2, and use FED2 `replace-with` FED1. In this case, all the crates that have FED2 dependencies will be resolved from FED1 as expected.

For information about source replacement, JFrog strongly recommends reading the official Cargo documentation.

#### Cargo Rewrite Code Example

```
[registry]
default = "artifactory-local"
global-credential-providers = ["cargo:token"][[registries.artifactory-local]]
index = "sparse+http://my-awesome.jfrog.io/artifactory/api/cargo/cargo-local/index/"

[source.artifactory-fed1]
registry = "sparse+http://my-awesome.jfrog.io/artifactory/api/cargo/cargo-fed1/index/" replace-with = "artifactory-fed2"

[source.artifactory-fed2]
registry = "sparse+http://my-awesome.jfrog.io/artifactory/api/cargo/cargo-fed2/index/" replace-with = "artifactory-fed1"

#artifactory-remote points to the github registry
[source.artifactory-remote]
registry = "sparse+http://my-awesome.jfrog.io/artifactory/api/cargo/cargo-remote/index/"

[source.crates-io]
replace-with = "artifactory-remote"
```

## 7.9 | Chef Cookbook Repositories

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.

#### Integration Benefits

[JFrog Artifactory and Chef Cookbook Repositories](#)

### 7.9.1 | Set Up a Chef Supermarket Repository

You can set up the following types of Chef supermarket repositories:

- Local Chef supermarkets
- Remote Chef supermarkets
- Virtual Chef supermarkets

#### 7.9.1.1 | Set Up a 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:

Chef Cookbook 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.

7.9.1.2 | Set Up a Remote Chef Supermarket

A Remote Repositories 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, follow the steps below:

1. In the **Administration** module, under **Repositories** | **Repositories** | **Remote**, click **New Remote Repository**.
2. Select **Chef** from the **Select Package Type** dialog.
3. In the New Repository dialog, set the **Repository Key** value, and specify the URL to the remote repository in the **URL** field as displayed below.

4. Click **Save & Finish**.

7.9.1.3 | Set up a 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 Repositories**, select **Chef** from the **Select Package Type** dialog, and select the underlying local and remote Chef repositories to include in the **Basic** settings tab.

#### 7.9.2 | Use 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 Cloud the URL would be:

`https://<server name>.jfrog.io/artifactory/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.

# JFrog Artifactory Documentation Displayed in the header

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`). You authenticate the request using your username and password or with your Artifactory API key :

## Setting the default Chef supermarket for Knife with credentials

```
knife[:supermarket_site] = 'http://admin:password@localhost:8081/artifactory/api/chef/chef-virtual'  
or  
knife[:supermarket_site] = 'http://admin:<APIKEY>@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

## 7.9.3 | Work with Artifactory and Chef 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.

## 7.9.4 | Publish Chef 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
```

## 7.9.5 | Use the Berkshelf Command Line

### Note

From version 6.1.0, Berkshelf supports authenticated access to Artifactory using an API Key. Previous versions of Berkshelf only supported Anonymous access to 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, first configure your Artifactory API Key using `config.rb`:

## Setting the Artifactory API key for use by Berkshelf

```
artifactory_api_key "<APIKEY>"
```

Then set the default supermarket in your Berksfile's Cookbook:

## Setting the default Chef supermarket for Berkshelf

```
source artifactory: '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

# JFrog Artifactory Documentation

## Displayed in the header

```
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 ([opscode] 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 ([opscode] 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 ([opscode] 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 ([opscode] 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 ([opscode] http://localhost:8081/artifactory/api/chef/chef-virtual/api/v1)
```

### 7.9.6 | View Individual Chef Cookbook Information

Artifactory lets you view selected metadata of a Chef Cookbook directly from the UI.

In the **Application** module, **Artifactory | Artifacts** tab, **Tree Browser**, drill down to select the `tgz/tar.gz` file you want to inspect. The metadata is displayed in the **Chef Info** tab.

### 7.9.7 | Search for 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.

## 7.10 | CocoaPods Repositories

Artifactory supports CocoaPods repositories on top of 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> or <https://github.com/CocoaPods/cdn.cocoapods.org>) through [Remote Repositories](#) which provide the usual proxy and caching functionality.
4. The ability to assign access privileges according to projects or development teams.

### 7.10.1 | Set Up a CocoaPods Repository

# JFrog Artifactory Documentation

## Displayed in the header

You can set up the following repository types:

- Local repositories
- Remote repositories
- Virtual repositories

### 7.10.1.1 | Set Up Local CocoaPods Repositories

To enable calculation of CocoaPods package metadata set **CocoaPods** to be the **Package Type** when you create your local CocoaPods repository.

#### Integration Benefits

##### JFrog Artifactory and CocoaPods Repositories

###### 7.10.1.1.1 | Deploy CocoaPods

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

### 7.10.1.2 | Set Up Remote CocoaPods 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 with a version lower than 5.1?

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
3. In the **CocoaPods Settings** section, select **GitHub** as the **Git Provider**, and leave the default **Registry URL** (<https://github.com/CocoaPods/Specs>).

# JFrog Artifactory Documentation Displayed in the header

Finally, click "Save & Finish"

## 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.

### 7.10.1.3 | Set Up Virtual CocoaPods Repositories

Starting from Artifactory version 7.84.x, Artifactory supports virtual CocoaPods repositories, only for repositories using CDN.

A **Virtual Repository** aggregates packages from both local and remote repositories that are included in the virtual repositories. This allows you to access packages that are hosted locally on local CocoaPods repositories, as well as remote packages that are proxied by remote CocoaPods repositories, and access all of them from a single URL defined for the virtual repository.

Virtual repositories can be very useful since users will continue to work with the virtual repository while the admin can manage the included repositories, and replace the default deployment target, while the changes will be transparent to the users.

To set up a virtual CocoaPods repository:

1. From the **Administration** module, select **Repositories** | **Repositories** | **Virtual**.
2. Click **New Virtual Repository** and select **CocoaPods** from the **Select Package Type** dialog.
3. Set the **Repository Key** value.
4. Select the underlying local and remote CocoaPods repositories to include under the **Repositories** section.

#### Include Only Repositories Using CDN

To be able to resolve artifacts from a CocoaPods virtual CDN repository, the Local and remote repositories that you include in your virtual repository must be CDN enabled. For more information, see [Identify Whether a CocoaPods Repository Uses CDN](#).

5. Enable CDN for this repository: for more information, see [Use CocoaPods CDN for Virtual Repositories](#)
6. (Optional) Configure your **Default Deployment Repository**, where the pods you upload to this virtual repository will be routed.

### 7.10.2 | Use the Pod Command Line (CDN)

#### 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 Cloud, the URL would be:

`https://<server name>.jfrog.io/artifactory/api/pods/<repository key>`

Artifactory has been updated to work seamlessly with the latest version of the CocoaPods client from version 0.39.0

You can configure the CocoaPods client with Artifactory in two ways:

- Using the CocoaPods CDN endpoint
- Using the `cocoapods-art` plugin

#### CocoaPods Git Indexing Deprecation

Starting July 2025, you will only be able to configure CocoaPods repositories using CocoaPods CDN, and the use of the `cocoapods-art` plugin will be deprecated. For more information, see [Deprecations in Process](#).

### 7.10.2.1 | Use CocoaPods CDN

CocoaPods CDN expedites the workflow by creating a static copy of the CocoaPods Specs repository, reducing the time required for adding repositories. For more information, see the [CocoaPods documentation](#). Note that if you are using CocoaPods CDN, you do not need to install the `cocoapods-art` plugin.

### Note

CocoaPods CDN is currently not supported for Smart Remote repositories.

To set up CDN on your repository, follow these guides:

- Use CocoaPods CDN for Local Repositories
- Use CocoaPods CDN for Remote Repositories
- Use CocoaPods CDN for Virtual Repositories

7.10.2.1.1 | [Use CocoaPods CDN for Local Repositories](#)

Starting from version 7.79.2, Artifactory supports using CocoaPods CDN for local repositories.

As a prerequisite for using CDN, re-index your repository manually: this is required to create the `podspec.json` file which CDN uses as the index. To reindex your repository, right-click the repository name in the **Artifacts** page in the JFrog Platform WebUI, and select **Recalculate Index** from the drop-down list, or run the Calculate CocoaPods Index REST API.

Then, to be able to resolve pods from the repository with CDN support, add the following line to your Podfile:

### Note

Make sure to replace the placeholders in **bold** with your own repository path.

```
source "<REPOSITORY_PATH>"
```

For example:

```
source "https://localhost:8080/artifactory/api/pods/coco-local"
```

After completing the pre-requisites, to use an Artifactory local repository with CocoaPods:

1. Log into CocoaPods as specified in your standard `.netrc` file:

### Note

Make sure to replace the placeholders in **bold** with your own JFrog machine, username, and password

```
machine <MACHINE>
localhost <YOUR_JFROG_USERNAME>
password <YOUR_JFROG_PASSWORD>
```

For example:

```
machine localhost
localhost admin
password AKCp2TfQM58F8FTkXo8qSJ8NymwJivmgefBqoJeEBQLSHCzsEH6Z2dmhS1siSxZTHoPPyUWJDhLGJ45kJKH2jHKWE7Sk
```

2. To add an Artifactory CDN repository, run the following command:

### Note

Make sure to replace the placeholders in **bold** with your own JFrog domain URL and repository.

```
pod repo add-cdn <REPOSITORY_NAME> "<YOUR_JFROG_DOMAIN>/api/pods/<REPOSITORY_PATH>"
```

For example:

```
pod repo add-cdn coco-local "https://localhost:8080/artifactory/api/pods/coco-local"
```

7.10.2.1.2 | [Use CocoaPods CDN for Remote Repositories](#)

To use an Artifactory remote repository with CocoaPods, do the following:

1. Log into CocoaPods as specified in your standard `.netrc` file:

# JFrog Artifactory Documentation

## Displayed in the header

### Note

Make sure to replace the placeholders in **bold** with your own JFrog machine, username, and password

```
machine <MACHINE>
localhost <YOUR_JFROG_USERNAME>
password <YOUR_JFROG_PASSWORD>
```

For example:

```
machine localhost
localhost admin
password AKCp2TfQM58FTkXo8qSJ8NymwJivmagefBqoJeEBQLSHCzsEH6Z2dmhS1si5xZTHoPPyUWJDhLGJ45kJKH2jHKWE7Sk
```

2. To be able to resolve pods from the repository with CDN support, add the following line to your Podfile:

### Note

Make sure to replace the placeholders in **bold** with your own repository path.

```
source "<REPOSITORY_PATH>"
```

For example:

```
source "https://john.jfrog.io/artifactory/api/pods/coco-virtual"
```

3. To add an Artifactory CDN repository, run the following command:

### Note

Make sure to replace the placeholders in **bold** with your own JFrog domain URL and repository.

```
pod repo add-cdn <REPOSITORY_NAME> "<YOUR_JFROG_DOMAIN>/api/pods/<REPOSITORY_PATH>"
```

For example:

```
pod repo add-cdn coco-remote "https://localhost:8080/artifactory/api/pods/coco-remote"
```

4. (Optional) By default, the **CDN URL** field points to the public registry <https://cdn.cocoapods.org/>. To use a different URL, define it in the **CDN URL** field, or modify the `podsCdnUrl` parameter via REST API.

7.10.2.1.3 | [Use CocoaPods CDN for Virtual Repositories](#)

Starting from version 7.84.x, Artifactory supports using CocoaPods virtual for CDN supported repositories.

### Include Only Repositories Using CDN

To be able to resolve artifacts from a CocoaPods virtual CDN repository, the Local and remote repositories that you include in your virtual repository must be CDN enabled. For more information, see [Identify Whether a CocoaPods Repository Uses CDN](#).

To use an Artifactory virtual repository with CocoaPods:

1. Log into CocoaPods as specified in your standard `.netrc` file:

# JFrog Artifactory Documentation

## Displayed in the header

### Note

Make sure to replace the placeholders in **bold** with your own JFrog machine, username, and password

```
machine <MACHINE>
localhost <YOUR_JFROG_USERNAME>
password <YOUR_JFROG_PASSWORD>
```

For example:

```
machine localhost
localhost admin
password AKCp2TfQM58FTkXo8qSJ8NymwJivmagedBqoJeEBQLSHCZusEH6Z2dmhS1si5xZTHoPPyUWJHDhLGJ45kJKH2jHKWE7Sk
```

2. To be able to resolve pods from the repository with CDN support, add the following line to your Podfile:

### Note

Make sure to replace the placeholders in **bold** with your own repository path.

```
source "<REPOSITORY_PATH>"
```

For example:

```
source "https://john.jfrog.io/artifactory/api/pods/coco-virtual"
```

3. To add an Artifactory CDN repository, run the following command:

### Note

Make sure to replace the placeholders in **bold** with your own JFrog domain URL and repository.

```
pod repo add-cdn <REPOSITORY_NAME> "<YOUR_JFROG_DOMAIN>/api/pods/<REPOSITORY_PATH>"
```

For example:

```
pod repo add-cdn coco-virtual "https://john.jfrog.io/artifactory/api/pods/coco-virtual"
```

### 7.10.2.1.4 | Identify Whether a CocoaPods Repository Uses CDN

Starting from version 7.8x.x, Artifactory supports using CocoaPods virtual for CDN supported repositories.

To identify whether a repository in your environment is CDN enabled or not:

1. From the Application module, go to **Artifactory > Artifacts**
2. Locate the repository you would like to check, and click the '>' symbol to expand its contents.
3. Look for a folder named **.pod/cdn/** and check if it contains any files. If the folder exists and has content in it, that means that it is indexed for CDN support.

### Regarding Remote Repositories

CDN-using remote repositories might not contain this folder if you have not resolved any artifacts from them.

To tell whether such a remote repository supports CDN, look for three or more levels of nested folders- if they exist within the remote repository, you can confidently assume that it is indexed for DCN support.

#### 7.10.2.2 | Use the cocoapods-art Plugin

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['COCOAPODS_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.

#### Synchronize 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
```

# JFrog Artifactory Documentation

## Displayed in the header

### 7.10.3 | Work with CocoaPods in 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 AKCp2TfQM8F8FTkXo8qSJ8NymwJivmgefbqoJeEBQLSHCZusEH6Z2dmhS1siSxZTHoPPyUW
```

#### Use an encrypted password

We recommend using an encrypted password instead of clear-text. For details, please refer to Centrally Secure Passwords.

### 7.10.4 | Clean 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:

#### Clean Pod Cache

```
pod cache clean
```

### 7.10.5 | Watch the CocoaPods Screencast

Watch this short screencast to learn how easy it is to host pods in Artifactory.



### 7.10.6 | Limitations of CocoaPods Repositories in Artifactory

CocoaPods CDN has the following limitations:

- Smart repositories are not currently supported for CocoaPods CDN
- Artifactory does not support dynamic variables in the podspec file.
- Artifactory supports all parameters mentioned in the [CocoaPods podspec documentation](#). Other syntax specifications have not been tested.
- The following commands are not supported:
  - \$ pod try
- CocoaPods local repositories currently doesn't support pods that have more than one podspec or podspec.json file present in the tar.gz.

## 7.11 | Conan Repositories

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.

For more details on building Conan packages and working with the Conan client, please refer to the [Conan documentation](#).

### Artifactory Community Edition for C/C++

Conan repositories are available in Artifactory CE.

> Learn more

#### Note

From Xray 3.21.2 and above, Xray can scan Conan packages, for more information see [Conan and C/C++ Support in Xray](#).

#### Tip

[Conan cheat sheet, the C/C++ Package Manager](#)

### 7.11.1 | Set Up a Conan Repository

You can set up the following repository types:

- Local Conan Repositories
- Remote Conan Repositories
- Virtual Conan Repositories

#### 7.11.1.1 | Local Conan Repositories

To enable calculation of C/C++ package metadata, from the **Administration** module, select **Repositories | Repositories | Local** and set **Conan** to be the **Package Type** when you create your local repository.

Make sure to also select `conan-default` as the repository layout.

#### Integration Benefits

[JFrog Artifactory and Conan Repositories](#)

#### 7.11.1.2 | Remote Conan Repositories

#### Deprecation Notice

JFrog Bintray is being sunset. Please refer this blog post for more detail.

A Remote Repository defined in Artifactory serves as a caching proxy for a registry managed at a remote URL such as <https://center.conan.io>, or even a Conan repository managed at a remote site by another instance of Artifactory.

Conan packages requested from a remote repository are cached on demand. You can remove Conan packages from the remote repository cache, however, you can not manually push Conan files to a remote Conan repository.

To define a remote repository to proxy a remote repository follow the steps below:

1. In the **Administration** module, under **Repositories | Repositories | Remote**, click **New Remote Repository**.
2. In the New Repository dialog, set the **Package Type** to **Conan**, set the **Repository Key** value, and specify the URL to the remote repository in the **URL** field as displayed below.

## Resolving Conan Remote Packages

To resolve Conan remote packages, aggregate the remote repository in a virtual repository as they cannot be resolved directly from the remote repositories.

### 7.11.1.3 | Virtual Conan Repositories

A Virtual Repositories defined in Artifactory aggregates Conan packages from both local and remote repositories that are included in the virtual repositories. 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 Conan repository follow these steps:

1. Create a new Virtual Repository, in the **Administration** module, under **Repositories | Repositories | Virtual**, click **New Virtual Repository** and set **Conan** as the **Package Type**.
2. Set the **Repository Key** value.
3. Select the underlying local and remote Conan repositories to include under the **Repositories** section.
4. You can optionally also configure your **Default Deployment Repository**

## JFrog Artifactory Documentation Displayed in the header

### 7.11.2 | Use 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.

Once you have created your Conan repository, select it in the Tree Browser view in the **Application** module, **Artifactory | Artifacts** tab, and click **Set Me Up** to see the code snippets you will need to use your repository as a source to install packages and as a target for export (upload packages).

#### Local vs. Remote

Don't let Conan terminology confuse you. For this integration, the Conan "Remote" is actually the Artifactory local repository you created for Conan packages.

In the sections below, <REMOTE> denotes the logical name you set with which the Conan client can identify the Conan local repository in Artifactory.

# JFrog Artifactory Documentation

## Displayed in the header

### Add Your Repository

#### Conan V2 Client

The instructions below apply for Conan client versions 2 and above- to use previous versions of the Conan client, see the [Conan documentation](#).

To use your local repository with Conan, you first need to add it as a Conan "Remote" to the client as follows:

#### Note

Make sure to replace your placeholders in **bold** with your own remote name, your JFrog domain URL, and Artifactory repository key

```
conan remote add <REMOTE> https://<YOUR_JFROG_DOMAIN>/api/conan/<REPOSITORY_KEY>
```

For example:

```
conan remote add my-artifactory-repo https://john.jfrog.io/api/conan/conan-local
```

#### Conan repositories must be prefixed with api/conan in the path

When accessing a Conan repository through Artifactory and using a command not copied from the Set Me Up instructions, 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 Cloud, the URL would be:

```
https://<server name>.jfrog.io/artifactory/api/conan/<repository key>
```

### Authenticate the Conan Client

#### Accessing Artifactory anonymously

If Artifactory is configured for anonymous access, you may skip authenticating the Conan client.

To authenticate the Conan client to Artifactory, log in using the following command:

#### Note

Make sure to replace your placeholders in **bold** with your own remote name, username, and password

```
conan remote login <REMOTE> <USERNAME> -p <PASSWORD>
```

For example:

```
conan remote login my-artifactory-repo johnf -p LEBoBAkwtvbwB6q
```

#### Allow Anonymous Access

Artifactory supports Conan repositories with **Allow Anonymous Access** enabled.

When **Allow Anonymous Access** is enabled, Artifactory will not query the Conan 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.



When set, Artifactory will first request authentication parameters from the Conan client before trying to access this repository.

### Install Dependencies

To install dependencies from Artifactory as defined in your `conanfile.txt` file, use the following command:

#### Note

Make sure to replace your placeholders in **bold** with your own remote name

```
conan install . -r <REMOTE>
```

For example:

```
conan install . -r my-artifactory-repo
```

### Upload Packages

To upload packages to your Artifactory local Conan repository, use the following command:

#### Note

Make sure to replace your placeholders in **bold** with your own remote name and pattern to the package, folder, or recipe that you want to upload. To upload a recipe, format it in the following way: `<NAME>/<VERSION>@<USER>/<CHANNEL>`

```
conan upload <PATTERN> -r <REMOTE>
```

For example:

# JFrog Artifactory Documentation

## Displayed in the header

```
conan upload "*" -r my-artifactory-repo
```

### 7.11.3 | View Individual Conan Package Information

Artifactory lets you view selected metadata of a Conan package directly from the UI.

In the **Application** module, **Artifactory | Artifacts** tab, **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.

If you select one of the packages, you get detailed Conan Package info including **Settings**, **Options** and dependencies ("Requires")

### 7.11.4 | Conan V2 Package Support

Conan server API v2 is supported and introduces an extension of the binary layout to support Conan Package revisions. Revisions allow you to change your artifacts while keeping the same Conan reference and is intended to achieve package immutability, by preventing data from being overwritten on the server.

This example shows the layout with "9999" as the <packageld>, "1" is the Recipe Revision, and "4" is the Package Revision.

```
user/lib/1.0/channel/1/package/9999/4/*
```

#### Revisions Support By Conan Client

When the Revision features is enabled, the Conan client searches for the latest revision in Artifactory per reference, unless specified otherwise by the user. It is not mandatory to upgrade your conan client version to use Artifactory 6.9 however if you want to work with revisions you need to download use Conan client 1.13 with the revision mode enabled.

The Conan client, by default, searches for the latest revision in Artifactory per reference, unless specified otherwise by the user.

#### 7.11.4.1 | Conan Package V1 Backward Compatibility

Artifactory 6.9.0 provides backward compatibility for packages created with Conan server API v1 by automatically migrating the Conan server API v1 binary layout to the new format.

Migration to Conan server API V2 starts after the upgrade process is complete (in all nodes in case of High Availability) and all Conan API endpoints are blocked and cannot be accessed.

After migration, the default revision for all Conan server API v1 packages is set to "0" for both Recipe revision and Package revision, and endpoints will be accessible again.

By default, two threads are dedicated for the migration job. Before upgrading from versions previous to Artifactory 6.9, you can modify this setting in the \$JFROG\_HOME/artifactory/etc/artifactory.system.properties file (\$JFROG\_HOME/artifactory/var/etc/artifactory/artifactory.system.properties if modifying 7.x).

Note that you can allocate more threads during the migration process but need to restart Artifactory.

```
artifactory.conan.v2.migration.job.queue.workers = 2 (default)
```

#### 7.11.4.2 | Conan Revision Indexing

Use the Conan client to deploy your packages. The Conan client will by default request the latest revision of a Conan reference requested.

Upon deployment using the Conan client, a .timestamp file is created under each revision root for each Recipe and Package revision root.

This file contains the epoch time of deployment (in milliseconds, for example, 1547984992855) and is created only when using the Conan client.

#### Only Use the Conan Client to deploy Conan Packages

Do not deploy the Conan packages in the UI or via REST API deployment to prevent index consistency and failed resolutions.

#### 7.11.4.3 | View Individual Conan V2 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.

#### 7.11.4.4 | View Conan Package Revisions

From Artifactory 6.9.0, Conan v2 is supported and introduces a new Conan format layout to support the Revisions attribute.

The following example shows a package with the default revision "0" for both Recipe and Package.

#### 7.11.5 | Conan and C/C++ Support in Xray

Xray can scan Conan packages deployed to Artifactory. Xray can also scan C/C++ dependencies as part of a build.

##### Note

Requires Artifactory version 7.17.4 and above.

#### 7.11.5.1 | Scan Conan Packages and Builds

##### Packages

Xray scans Conan packages the same way it scans other package types. Xray data will only be displayed for the `conanmanifest.txt` file. An optional vendor field can be added in the Conan recipe file to prevent false positives.

##### Builds

Conan artifacts and dependencies can be provided as part of the BuildInfo using the `conan_build_info` command.

#### 7.11.5.2 | Scan C/C++ Builds

Xray supports scanning C/C++ packages as build-dependencies only. The following steps are required:

- Create a build-info listing all the C/C++ packages you want to scan. Refer to the [Build-info Creation example](#).
- Upload the build to Artifactory, and perform an Xray scan.

For more information, see the [Build Upload REST API](#).

##### Note

This process requires creating and uploading C/C++ build-info manually in accordance with the build-info schema. It also requires the listing of all the C/C++ libraries to be scanned. For each component you need to provide name and version; vendor is optional.

In the BuildInfo, do the following:

1. Specify a `cpp` module, and set `cpp` as the `modules` type.
2. In the `dependencies` section, list all of the `cpp` components of your build. Each `cpp` component must contain:

- **Sha1**
- **ID:** Consists of the component's vendor, name and version, in the form: "[`vendor`] : `name` : `version`". Please note that the '`vendor`' field is optional.

##### C/C++ Info Example:

```
{  
  "version": "1.0.1",  
  "name": "MyBuildName",  
  "number": "42",  
  "type": "GENERIC",  
  "started": "2021-01-19T15:47:52.000Z",  
  "dependencies": [  
    {  
      "id": "com.example:mylib:1.0.1"  
    }  
  ]  
}
```

```
"buildAgent": {
    "name": "Private builder",
    "version": "1.0"
},
"modules": [
    {
        "id": "<MODULE-ID>",
        "type": "cpp",
        "dependencies": [
            {
                "sha1": "<SHA1>",
                "md5": "<MD5>",
                "id": "<vendor1>:<name1>:<version1>",
                "type": "cpp"
            },
            {
                "sha1": "<SHA1>",
                "md5": "<MD5>",
                "id": "<vendor2>:<name2>:<version2>",
                "type": "cpp"
            }
        ]
    }
}
```

### Build-info Creation Example :

```
# Choose between A or B or C (depending where your dependencies are located) :
# # A. add Build info dependencies located on the local disk
# ##### jfrog rt bad myLibs/ cpp_build 1
# # B. add Build info dependencies located in Artifactory
# ##### jfrog rt bad mcy-cpp-deps/ --from-rt=true cpp_build 1
# # C. add Build info dependencies by downloading them from Artifactory
# ##### jfrog rt dl mcy-cpp-deps/ cpp_build 1

# generate Build info and save it as JSON file
jfrog rt bp --dry-run=true cpp_build 1 > build_info.json

# the following command will :
# 1. add type=cpp to the module
# 2. add type=cpp for each dependency
# 3. update the component id for each dependency
jq '.modules[] += {"type":"cpp"}' build_info.json |\
jq '.modules[].dependencies[] += {"type":"cpp"}' |\
jq '(.modules[].dependencies[] | select(.id == "Poco.dll") | .id) |= "poco:1.8.0"' |\
jq '(.modules[].dependencies[] | select(.id == "libcurl.dll") | .id) |= "haxx:libcurl:7.70.0"' |\
jq '(.modules[].dependencies[] | select(.id == "sqlite.dll") | .id) |= "sqlite:3.15.1"' |\
jq '(.modules[].dependencies[] | select(.id == "zlib.dll") | .id) |= "zlib:1.2.0"' > build_info_xray.json

# upload build info
jfrog rt cl ...
```

## 7.12 | Conda Repositories

Artifactory natively supports Conda repositories for Python, R, Ruby, Lua, Scala, Java, JavaScript, C/C++, FORTRAN and additional programming languages, giving you full control of your deployment and resolution process of Conda packages.

### Minimal supported Conda client version

Artifactory supports Conda Client version 4.12.0 and above. We recommend using the latest version.

Conda repositories offer the following benefits:

- Secure and private local Conda repositories with fine-grained access control.
- The ability to proxy remote Conda resources and cache downloaded Conda packages to keep you independent of the network and the remote resource.
- Virtual Conda repositories that support a single URL through which to manage the resolution and deployment of all your Conda packages.
- Metadata calculation of the Conda packages hosted in the Artifactory local repositories.
- Version management: Archiving older versions of the packages uploaded to local repositories.

### 7.12.1 | Set Up a Conda Repository

You can set up the following repository types:

- Local Conda Repositories
- Remote Conda Repositories
- Virtual Conda Repositories

#### 7.12.1.1 | Local Conda Repositories

To enable calculation of Conda metadata, in the **Administration** module, go to **Repositories** | **Repositories** | **Local** and select **Conda** as the **Package Type** when you create your local repository.

### 7.12.1.1 | Local Conda Repository Layout

The local Conda repository in Artifactory gives you the flexibility of deploying your packages in a layout of your choice. When deploying Conda binaries into nested paths, it is important to ensure that the channel URL inside your `.condarc` file correctly reflects the path of the packages. The Conda client automatically appends the host machine platform as a subdirectory in the channel URL. When you deploy these packages, you need to meet this requirement and upload your packages into the relevant subdirectories. For example, consider the following valid package path in Artifactory:

```
conda-local/osx-64/my-conda-package.tar.bz2
```

Based on this layout, the corresponding channel URL in your `.condarc` file is:

```
<YOUR_SERVER_URL>/artifactory/api/conda/conda-local
```

In the above example, Conda will be appending the platform automatically (i.e osx-64).

Another example shows a more detailed deployment path such as:

```
conda-local/my/own/layout/osx-64/my-conda-package.tar.bz2
```

In this example, the channel URL should be set as follows:

```
<YOUR_SERVER_URL>/artifactory/api/conda/conda-local/my/own/layout
```

### Best Practice for Scaling Up

The Conda repository metadata is maintained on the package level, meaning that the parent directory of a Conda package is also the parent of the Conda `repodata.json` metadata file. To scale with maximum efficiency, refrain from deploying large amounts of packages into a single parent directory when possible. To enable Artifactory metadata calculation processes to provide maximal performance, try to plan your local repository layouts in a way that supports modularity by avoiding directories with large amounts of artifacts as their direct children.

### 7.12.1.2 | Remote Conda Repositories

You can create a Conda remote repository to proxy and cache remote repositories or other Artifactory instances.

Note that the index files for remote Conda repositories are stored and renewed according to the Retrieval Cache Period setting on your remote repository.

### 7.12.1.3 | Virtual Conda Repositories

A virtual repository in Artifactory aggregates packages from both local and remote repositories allowing you to access both locally hosted Conda packages and remote proxied Conda libraries from a single URL defined for the virtual repository.

To create a virtual Conda repository, set **Conda** as the **Package Type**, and select the underlying local and remote Conda repositories to include under the **Repositories** section.

### Virtual Repository Metadata

Artifactory maintains your aggregated virtual repository metadata in a clever way that keeps your metadata up-to-date by reflecting changes in the aggregated **local** repositories in real-time. For aggregated **remote** repositories, the virtual repository metadata is renewed on-demand, in minimal intervals of 10 minutes by default. The renewal period is controlled using the Retrieval Cache Period parameter of your virtual repository.

#### 7.12.2 | Resolve Conda Packages

You can resolve Conda packages in two ways:

- [Resolve Conda Packages Using the UI](#)
- [Resolve Conda Packages Using the Conda Client](#)

##### 7.12.2.1 | Resolve Conda Packages Using the UI

When a Conda repository is selected in the Artifacts Tree Browser, click **Set Me Up** to view the code snippets you can use to publish a Conda package or to configure your Conda client to resolve artifacts using the selected repository.

##### 7.12.2.2 | Resolve Conda Packages Using the Conda Client

1. Perform the process of settings up your `.condarc` file according to the instructions in the **Set Me Up** page for Conda.

2. Install a package from your Artifactory Conda repository:

```
conda install <PACKAGE_NAME>
```

3. Install a package from a specific sub-channel inside your Conda repository:

```
conda install -c <CHANNEL_NAME> <PACKAGE_NAME>
```

4. Search for a package in your Artifactory Conda repository:

```
conda search <PACKAGE_NAME>
```

#### 7.12.3 | Deploy Conda Packages

You can deploy packages to a local or virtual Conda repository using the **Deploy** feature in the UI or using an HTTP client of your choice.

### Metadata Updates

The Conda metadata is automatically calculated and updated when adding, removing, copying or moving Conda packages. The calculation is only invoked after a package-related action is completed.

It may sometimes take up to 30 seconds to complete as the process is asynchronous and its performance depends on the overall system load.

# JFrog Artifactory Documentation

## Displayed in the header

Although rarely required, you may wish to invoke metadata calculation on the entire repository. This can be done using the **Recalculate Index** option after you right-click the repository in the Tree Browser, or via the REST API.

### Set the Default Conda Deployment Repository

To deploy Conda packages to a virtual Conda repository, make sure you have set the **Default Deployment Repository**.

### Deploying a Package Using the UI

You can drag and drop, or select a Conda package to upload in Deploy in the UI.

#### Deploy a Conda Source Package

When deploying sources, the Target Path is automatically displayed and we recommend not changing this path. Changing the 'src/contrib' path will result in Artifactory not identifying the package as a Conda package since Artifactory will not be able to index it.

### Deploy a Conda Package Using cURL

To deploy your package to an Artifactory repository you can either use the Artifactory web UI, or upload the package using an HTTP client such as cURL:

#### Deploy source package

```
curl -XPUT ${USERNAME}:${PASSWORD} "http://localhost:8080/artifactory/conda-local/" -T my-package-1.0.0.tar.bz2
```

#### 7.12.4 | View Individual Conda Package Information

Artifactory supports viewing selected Conda package metadata directly from the UI.

In the **Artifact Repository Browser**, select your virtual Conda repository and scroll down to find and select the package you want to inspect. The metadata is displayed in the **Conda Info** tab.

#### 7.12.5 | Reindex a Conda Repository

You can trigger asynchronous reindexing of a local Conda repository either through the UI or using the REST API.

Through the UI, select your Conda repository in the **Repositories List** and select **Recalculate Index**. This requires Admin permissions.

To reindex a Conda repository through the REST API, please refer to Calculate Conda Repository Metadata.

#### 7.12.6 | Tune Conda Metadata Worker Threads

##### Conda Metadata Workers

Conda calculates metadata asynchronously based on repository storage events. The number of total worker threads that handle metadata calculation in parallel (specifically for Conda tasks) defaults to 5. In larger scales, you may modify this parameter by editing your `$JFROG_HOME/artifactory/var/etc/artifactory/artifactory.system.properties` file and adding the following parameter:

```
artifactory.conda.metadata.calculation.workers=<NUMBER_OF_WORKERS>
```

### 7.13 | CRAN Repositories

Artifactory natively supports CRAN repositories for the R language, giving you full control of your deployment and resolving process of CRAN packages. The Comprehensive R Archive Network (CRAN) is a collection of sites that carry identical material, consisting of R distributions, contributed extensions, R documentation, and binaries. R is a programming language and free software environment dedicated to statistical computing and graphics that is supported by the R Foundation for Statistical Computing. The R language is widely used for developing statistical software and data analysis by data miners and statisticians.

CRAN repositories in Artifactory offer the following benefits:

- Secure and private local CRAN repositories with fine-grained access control.
- The ability to proxy remote CRAN resources and cache downloaded CRAN packages to keep you independent of the network and the remote resource.
- Virtual CRAN repositories that support a single URL through which to manage the resolution and deployment of all your CRAN packages.
- Metadata calculation of the CRAN packages hosted in the Artifactory local repositories.
- Version management: Archiving older versions of the packages uploaded to local repositories.
- Manage source and binaries.

### CRAN version support

Artifactory supports CRAN version 3.4.0 and above.

#### 7.13.1 | Set Up a CRAN Repository

You can set up the following CRAN repository types:

- Local CRAN Repositories
- Remote CRAN Repositories
- Virtual CRAN Repositories

##### 7.13.1.1 | Set Up Local CRAN Repositories

To enable calculation of CRAN metadata, in the Administration module, go to **Repositories| Repositories | Local** and select **CRAN** as the **Package Type** when you create your local repository.

#### Local CRAN Repository Layout

You need to maintain a specific path structure to manage the CRAN packages that are uploaded to CRAN local repositories.

CRAN packages are uploaded to the following locations:

- Source packages are automatically uploaded by default to the relative path: `src/contrib`. For example: `src/contrib/ArtifactoryRDS_0.1.0.tar.gz`.
- Binary packages are uploaded to a relative path according to the distribution and R version. For example: `/myfirstpkg_1.2.tgz`.

Artifactory will find your packages by performing a property search causing the folder hierarchy not to have an impact on performance.

#### Placing source packages in the recommended path

When uploading a CRAN package via the UI, the default deploy path is not enforced but is recommended since it allows Artifactory to manage the CRAN packages. Uploading the packages to a different path will cause the packages not to be identified as CRAN packages.

##### 7.13.1.2 | Set Up Remote CRAN Repositories

You can create CRAN remote repository to proxy and cache remote repositories or other Artifactory instances.

Note that the index files for remote CRAN repositories are stored and renewed according to the Retrieval Cache Period setting.

##### 7.13.1.3 | Set Up Virtual CRAN Repositories

A Virtual Repository in Artifactory aggregates packages from both local and remote repositories. This allows you to access both locally hosted CRAN packages and remote proxied CRAN libraries from a single URL defined for the virtual repository.

To create a virtual CRAN repository, set **CRAN** as the **Package Type**, and select the underlying local and remote CRAN repositories to include under the **Repositories** section.

### 7.13.2 | Resolve CRAN Packages

There are two ways to resolve CRAN Packages:

- [Using the UI](#)
- [Using the R command line](#)

#### 7.13.2.1 | Resolve CRAN Packages Using the UI

When a CRAN repository is selected in the Artifacts module Tree Browser, click **Set Me Up** to view the code snippets you can use to publish a CRAN package or to configure your R client to resolve artifacts using the selected repository.

#### 7.13.2.2 | Resolve CRAN Packages Using the R Command Line

1. Run the **Set Me Up** for CRAN.
2. To switch from the current repository to a different resolution CRAN repository.

```
setRepositories()
```

3. View all the available packages for the selected CRAN repository.

```
available.packages()
```

4. Install a package from the CRAN repository.

```
install.packages() - select from a list, on supported clients  
install.packages("package") - install by package name
```

#### Note

Please note that the CRAN client only allows basic authentication in the URL section, using either a password or a reference token.

### 7.13.3 | Deploy CRAN Packages

You can deploy packages to a local or virtual CRAN repository using the **Deploy** feature in the UI or using a POST request. To learn more, see:

- [Deploy CRAN Packages Using the UI](#)
- [Deploy CRAN Packages Using cURL](#)

#### Metadata Updates

The CRAN metadata is automatically calculated and updated when adding, removing, copying or moving CRAN packages. The calculation is only invoked after a package-related action is completed.

It may sometimes take up to 30 seconds to complete as the process is asynchronous and its performance depends on the overall system load.

You can also invoke metadata calculation on the entire repository by selecting **Reindex Packages**.

#### Set the Default Deployment CRAN Repository

To deploy CRAN packages to a virtual CRAN repository, make sure you have set the **Default Deployment Repository**.

# JFrog Artifactory Documentation

## Displayed in the header

### 7.13.3.1 | Deploy a CRAN Package Using the UI

You can drag and drop, or select a CRAN package to upload in Deploy in the UI. Artifactory will identify if it's a source or binary package.

Artifactory supports two types of packages: binaries and sources. They are treated differently in terms of the deployment in the UI. To learn more, see:

- Deploy a Source CRAN Package
- Deploy a Binary CRAN Package

#### 7.13.3.1.1 | Deploy a Binary CRAN Package

In binary deploy, you'll need to fill the CRAN Artifact section.

In the CRAN Artifact section, configure these fields when deploying the CRAN packages. It is mandatory to set these fields and are used to create the destination path of the deployed binary package.

- Distribution: Specifies the operating system.
- R Version: Indicates the R version used.

#### Target Path

The Target path is updated **after** the file is deployed and there is no need to change it

#### 7.13.3.1.2 | Deploy a Source CRAN Package

When deploying sources deploy, the Target Path is automatically displayed and we recommend not changing this path. Changing the `src/contrib` path will result in Artifactory not identifying the package as a CRAN package since Artifactory will not be able to index it.

#### Target Path

The Target path is automatically updated and changing it could make Artifactory not invoke the metadata calculation, and this package to not get indexed.

#### 7.13.3.2 | Deploy a CRAN Package Using cURL

##### Deploy source package

```
curl -XPOST "http://localhost:8080/artifactory/api/cran/cran-local/sources" -T package_1.0.tar.gz
```

##### Deploy binary package

```
curl -XPOST "http://localhost:8080/artifactory/api/cran/cran-local/binaries?distribution=macosx/el-capitan&rVersion=3.5" -T package_1.0.tgz
curl -XPOST "http://localhost:8080/artifactory/api/cran/cran-local/binaries?distribution=windows&rVersion=3.5" -T package_1.0.zip
```

When deploying a CRAN binary package, you need to specify the distribution and R version as before.

When deploying directly (PUT request to a specific path), make sure the target path is a valid CRAN path:

- /src/contrib for sources
- /bin/{distribution}/contrib/{r-version} for binaries.

Deploying a package to a different path will not identify the package as CRAN packages, and will not invoke the metadata indexing.

#### 7.13.4 | Apply the CRAN Official Specification to Local CRAN Repositories

From Artifactory version 7.41.1, you can set the artifacts stored in local CRAN repositories to confine to the official CRAN spec requirements.

Set the following system property in the \$JFROG\_HOME/artifactory/var/etc/artifactory/artifactory.system.properties file to save your CRAN archives in the correct hierarchy.

"artifactory.cran.archiveMover.enabled"

You can move your existing Archives to the correct path using the following Move CRAN Archives REST API. Requires the artifactory.cran.archiveMover.enabled property to be enabled.

Restart Artifactory after setting the property for the change to take place.

#### 7.13.5 | View Individual CRAN Package Information

Artifactory lets you view selected metadata of a CRAN package directly from the UI.

In the [Artifact Repository Browser](#), select your virtual CRAN repository and scroll down to find and select the package you want to inspect. The metadata is displayed in the [CRAN Info](#) tab.

#### 7.13.6 | Reindex a CRAN Repository

You can trigger an asynchronous reindexing of a local CRAN repository either through the UI or using the REST API.

Through the UI, select your CRAN repository in the Tree Browser and select Recalculate Index from the right-click menu as shown below (requires Admin privileges)

To reindex a CRAN repository through the REST API, please refer to Calculate CRAN Repository Metadata.

#### 7.14 | Debian Repositories

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:

# JFrog Artifactory Documentation

## Displayed in the header

- The ability to provision Debian packages from Artifactory to a Debian client from local and remote repositories.
- Calculation of Metadata for Debian packages hosted in Artifactory's local repositories.
- Access to remote Debian resources (such as us.archive.ubuntu.com) through Remote Repositories which provide the usual proxy and caching functionality.
- Providing GPG signatures that can be used by Debian clients to verify packages.
- Complete management of GPG signatures using the Artifactory UI and the REST API.
- Support for deploying Debian Snapshots.

### 7.14.1 | Set Up a Debian Repository

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.

#### 7.14.1.1 | Set Up Local Debian Repositories

To enable calculation of Debian metadata, in the **Administration** module, go to **Repositories| Repositories | Local** and select **Debian** as the **Package Type** when you create your local repository.

If you are using Debian with a *Trivial* layout, in the **Debian Settings** section, set the **Trivial Layout** checkbox.

Under **Optional Index Compression Formats**, click on **Select compression formats** and from the drop-down list select the index file formats you would like to create in addition to the default Gzip (.gzip extension), which is created for every Debian repository and cannot be disabled. Additional index file formats can also be created using the Local Repository REST API.

To allow indexing of debug symbols, select the **Enable indexing with debug symbols (.ddeb)** field.

#### 7.14.1.1.1 | Deploy a Debian package using the UI

To deploy a Debian package to Artifactory, in the Artifacts Repository Browser, click **Deploy**.

Select your Debian repository as the **Target Repository**, upload the file you want to deploy.

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.

#### 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**.

#### Tip

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:

# JFrog Artifactory Documentation

## Displayed in the header

### Deploying a package using Matrix Parameters

```
PUT "http://SERVER_HOSTNAME:8081/artifactory/{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:8081/artifactory/debian-local/pool/libatk1.0_i386.deb;deb.distribution=wheezy;deb.component=main;deb.architecture=i386"
```

#### 7.14.1.1.3 | Set the Target Path for Debian

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]
```

Property	Description
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 <b>pool</b> folder in accordance with the Debian convention.
distribution	The value to assign to the deb.distribution property used to specify the Debian package distribution
component	The value to assign to the deb.component property used to specify the Debian package component name
architecture	The value to assign to the deb.architecture 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.

#### 7.14.1.1.4 | Specify multiple Debian 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:8081/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"
```

#### 7.14.1.1.5 | Debian Artifact Metadata

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.

#### 7.14.1.1.6 | Use Debian 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:

# JFrog Artifactory Documentation

## Displayed in the header

- 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

- 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.

#### 7.14.1.2 | Set Up Remote Debian 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.

Note that the index files for remote Debian repositories (including the sources index) are stored and renewed according to the **Retrieval Cache Period** setting.

#### 7.14.1.2.1 | Calculate Debian Coordinates

You can extract the component/distribution/architecture coordinates from a remote repository and assign them as properties on the cached packages.

To do this, right click on the relevant remote Debian repository and select **Calculate Debian Coordinates**. You can also use the Calculate Cached Remote Debian Repository Coordinates REST API.

### Note

This process may take some time for remote repositories with many packages. It is recommended running it only when needed. For example, before copying packages to a local repository.

The following executions will only calculate the newly added packages to the cache.

#### 7.14.1.3 | Set Up Virtual Debian Repositories

A Virtual Repository defined in Artifactory aggregates packages from both local and remote repositories.

This allows you to access both locally hosted Debian packages and remote proxied Debian repositories from a single URL defined for the virtual repository.

To define a virtual Debian repository, create a virtual repository, set the **Package Type** to be **Debian**, and select the underlying local and remote Debian repositories to include in the **Basic** settingstab.

##### Deprecated Trivial layout not supported

Only repositories with an Automatic Layout can be included in a virtual repositories. A deprecated **Trivial** layout is not supported for virtual repositories.

##### Indexed Remote Architectures

When creating a new virtual repository, the **Indexed Remote Architectures** field specifies the architectures which will be indexed for the included remote repositories.

Specifying these architectures will speed up Artifactory's initial metadata indexing process. The default architecture values are `amd64` and `i386`.

Set the **Optional Index Compression Formats** with the index file formats you would like to create in addition to the default Gzip (.gzip extension).

#### 7.14.2 | Sign Debian Metadata

Artifactory supports signing Debian repository metadata (not packages) using a GPG key. This process will create a signed Release file named `Release.gpg`, which 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, see [Managing Signing Keys](#).

### 7.14.3 | Work with Debian Snapshots

This section contains the following topics:

- [Create Debian Snapshots](#)
- [Rules and Guidelines for Working With Debian Snapshots](#)
- [Resolve Debian Snapshots](#)

#### 7.14.3.1 | Create Debian Snapshots

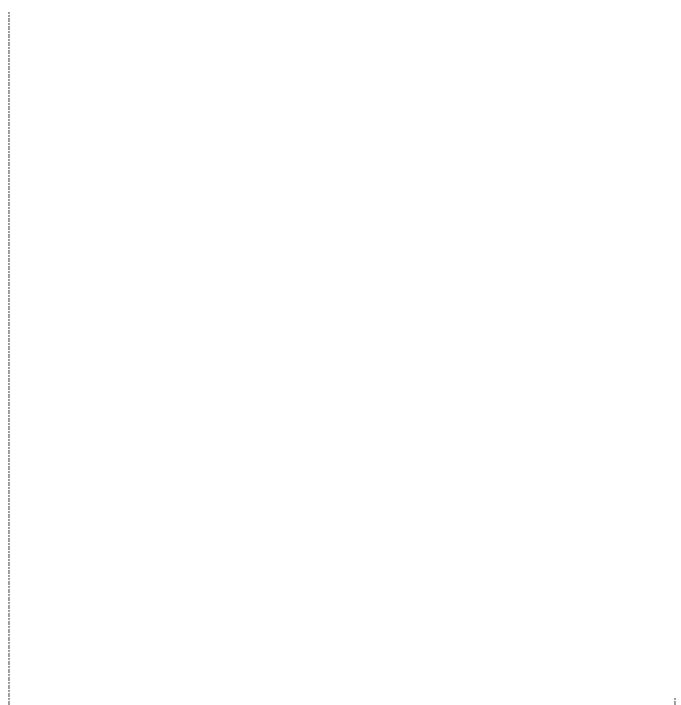
From Artifactory 7.41.3, Debian repositories include support for Debian Snapshots. A Debian Snapshot contains metadata of a fixed state of a local, remote or virtual Debian repository. Snapshots are immutable and contain a metadata folder for each specific distribution, including all metadata files, like RELEASES AND PACKAGES, and not the actual packages.

Snapshots can be used in the following scenarios:

- As backups, allowing you to easily fall back to previous versions in case of package corruption due to dependency changes.
- For release purposes, whereby the tested Packages file can be immutably saved and served.

In Artifactory, the Debian snapshots are saved by default under the \$repoKey/snapshots/\$tag subfolder within the Debian Repository and are created using a dedicated Create Debian Snapshot REST API.

In the following example, a Debian snapshot named "202203141800" is saved under the "deb-snapshot" local repository, containing "focal" distribution metadata.



#### 7.14.3.2 | Rules and Guidelines for Working With Debian Snapshots

- To work with Debian Snapshots in Artifactory, you need to configure the sources file of your Debian client to point to the following path:  
`artUrl/artifactory/api/deb/$repoKey/snapshots/$tag`.
- The root of the snapshot sub-repository is `$repoKey/snapshots/$tag`.
- All the metadata files of the `$srcRepo/dists/$distribution`, including the release and packages file, are copied to the snapshot folder.
- In the folder, a property called `deb.snapshot.source` is created pointing to the source repository.
- You need to use `/artifactory/api/deb/$repoKey/snapshots/$tag`, whereby the tag is a user-defined name for the snapshot.
- Deleting a snapshot does not delete the related Debian Package files as the. Debian Snapshots sub-repositories does not contain the packages, but only redirect to them.

To create Debian Snapshot, see the Create Debian Snapshot REST API.

#### 7.14.3.3 | Resolve Debian Snapshots

To work with Debian Snapshots in Artifactory, you need to configure the sources file of your Debian client to point to the following path:  
`artUrl/artifactory/api/deb/$repoKey/snapshots/$tag`.

### 7.14.4 | Add MD5 Checksum to the Debian 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.

### 7.14.5 | Configure Authenticated Access to Debian 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@SERVER_HOSTNAME:8081/artifactory/{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) - created by default for every Debian repository and cannot be disabled.
- Bzip2 (.bz2 file extension)
- LZMA (.lzma extension)
- XZ (.xz extension)

### 7.14.6 | Acquire Debian Packages by Hash

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:

Parameter	Description
artifactory.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
artifactory.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

### 7.14.7 | Use Debian InRelease Metadata Files

From version 7.4, Artifactory supports Debian InRelease metadata files.

Artifactory will produce an InRelease metadata file in the repository when working with GPG signing. Downloading a Debian package from Artifactory will now be faster as the client will only download the InRelease file without downloading the Release and Release.gpg files that are heavier.

#### Debian Release Files Configuration

The Release files default values for the Label and Origin parameters is Artifactory. From Artifactory version 6.23.3 or 7.11.1, it is possible to change these values.

To do so, set the deb.release.origin and/or deb.release.label custom properties to the desired value on the root repository level.

### 7.14.8 | Debian 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

7.14.9 | [Watch the Debian Screencast](#)



### 7.14.10 | Limitations of Debian Repositories in Artifactory

- Artifactory only supports binary Debian packages. Source packages are not supported.

## 7.15 | 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.

### Supported Docker Versions

Artifactory supports Docker client versions 19.03.15 and above.

### 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.

### OCI Support

Artifactory is OCI compliant and supports OCI clients, enabling you to deploy and resolve OCI images in Docker Registries. The OCI client Singularity is not supported.

### Docker Buildx Support

Artifactory supports Docker Buildx, allowing you to easily build and push multi-architecture images using the Docker buildx CLI. For more information, see [Push Images in Bulk Using the Docker Buildx CLI](#).

### Confidently Promoting Images to Production

Artifactory lets you promote Docker images, as immutable, stable binaries, through the quality gates all the way to production. For more information, see the [Promote Docker Image REST API](#).

### Unlimited Docker Hub access

Artifactory provides you with unlimited, high-performant access to Docker Hub and to Docker Official Images to simplify cloud-native application development, without Docker Hub image-pull limits. This allows you to streamline, automate and simplify the way DevOps teams work.

\* Available to SaaS cloud JFrog Platform subscribers, including those with AWS, GCP or Azure's free tier.

### 7.15.1 | Docker 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.

### Integration Benefits Docker Registry

### 7.15.2 | Set Up a Docker Repository

Artifactory supports three types of repositories when working with Docker:

# JFrog Artifactory Documentation

## Displayed in the header

- **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 the users knowing, and continue to work with the same end point.

\*\*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.

### Do not use underscores when naming Docker repositories

Due to a limitation in the Docker client, underscores are not permitted in Docker registry names. Therefore, when naming Artifactory Docker repositories, you should not use an underscore. For example, the Docker client will not be able to communicate with a repository named `test_docker_repo`, however, it **will** work with a repository named `test.docker.repo`.

#### 7.15.2.1 | 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:

1. From the **Administration** module, select **Repositories | Repositories | Local**.
2. Click **New Local Repository** and select **Docker** from the **Select Package Type** dialog.
3. Set the **Repository Key**, and in the Docker Settings section, select **V2** as the Docker API version.
4. 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.

#### 7.15.2.2 | 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.

### Action Required to Prevent Docker Remote Registry Restrictions

In lieu of the latest Docker remote repository limitations enforced by Docker, anonymous users on Self-Hosted platforms will be blocked when reaching the download rate limit of 100 pulls per six hours. To prevent this from happening, you are required to authenticate with Docker Hub, by setting your Docker account user and password in your **Remote Docker Repositories**.

To define a remote repository to proxy a remote Docker registry follow the steps below:

1. From the **Administration** module, select **Repositories | Repositories | Remote**.
2. Click **New Remote Repository** and select **Docker** from the **Select Package Type** dialog.
3. In the Basic tab, set the **Repository Key** value, and specify the URL to the remote registry in the **URL** field.

If you are proxying the Docker Hub, use <https://registry-1.docker.io/> as the URL, and make sure the **Enable Token Authentication** checkbox is checked (these are the default settings).

4. To use your Docker account type, you need to authenticate the Docker Hub pull requests, by setting your user and password in your basic Docker repository.

5. Click the Advanced tab to configure the Advanced Docker Repository settings you can enable Foreign Layers Caching to allow Artifactory to download foreign layers to a Docker remote repository.



6. Select the **Enable Foreign Layers Caching** checkbox to allow Artifactory to download foreign layers to a Docker remote repository.
7. You have an option to apply Patterns Allow List by setting Include patterns to match external URLs when trying to download foreign layers.  
Specify an Allow List of Ant-style path expressions that specify where foreign layers may be downloaded from. Supported expressions include (\*, \*\*, ?).  
By default, this field is set to \*\* which means that foreign layers may be downloaded from any external source.  
For example, specifying \*/ like in [github.com/](https://github.com/)\*\* will only allow downloading foreign layers from a github.com host.
8. To configure the Network settings, see Network Settings.
9. Click **Save and Finish**.

#### 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>`

# JFrog Artifactory Documentation

## Displayed in the header

### 7.15.2.3 | Virtual Docker Repositories

Artifactory supports virtual Docker Repositories. A Virtual Repositories 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. From the Administration module, select **Repositories** | **Repositories** | **Virtual**.
2. Click **New Virtual Repository** and select **Docker** from the **Select Package Type** dialog.
3. Set the **Repository Key** value.
4. Select the underlying local and remote Docker repositories to include under the **Repositories** section.
5. 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.

### Resolve Latest Docker Image

To set your virtual Docker repository to pull Docker images according to their modification time, enable *Resolve Docker Tags By Latest Timestamp*. This is useful in scenarios where two or more aggregated repositories contain the same tag name. For example, `busybox:1.1`.

When enabled, instead of fetching the image that is positioned higher in the resolution order in the virtual repository, Artifactory will return the Docker image last deployed to one of the aggregated repositories in the Virtual repository. Artifactory will first try to fetch the tag from the Local repositories according to the modification time, if not found, it will continue to try to fetch the image from the Remote repositories according to the resolution order.

This functionality is useful for multi-site environments where you create the same image on two different instances.

### REST API

This can also be configured by setting the `resolveDockerTagsByTimestamp` parameter to true (false by default) when creating a new repository using REST API.

### 7.15.2.4 | Docker Reverse Proxy Settings

Artifactory supports access to Docker registries either through a reverse proxy (using the subdomain method or through port bindings), or using direct access.

When accessing through a reverse proxy, if you are using the Artifactory Reverse Proxy configuration generator you can configure a Docker repository's reverse proxy settings under the **Advanced** settings tab.

### 7.15.3 | Promote 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.

#### Note

Starting from Artifactory version 7.94.1, Retagging a docker image using docker promotion enforces tag validation according to the OCI specification by default. To avoid using tag validation, set the `artifactory.docker.filter.digests.from.tags.list.enabled` parameter to `false` in your system configuration file.

Promotion can be triggered using the following endpoint with cURL:

```
POST api/docker/<repoKey>/v2/promote
{
    "targetRepo" : "<targetRepo>",
    "dockerRepository" : "<dockerRepository>",
    "tag" : "<tag>",
    "targetTag" : "<tag>",
    "copy": <true | false>
}
```

where:

Parameter	Description
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 its tags from `docker-local` to `docker-prod` using cURL would be:

```
curl -i -uadmin:password -X POST "https://artprod.company.com/api/docker/<repoKey>/v2/promote" -H "Content-Type: application/json" -d '{"targetRepo":"docker-prod","dockerRepository":"jfrog/ubuntu"}'
```

```
https://artprod.company.com/api/docker/ <repoKey>/v2/promote
```

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/api/docker/docker-local/v2/promote" -H "Content-Type: application/json" -d '{"targetRepo":"docker-prod","dockerRepository":"jfrog/ubuntu", "tag" : "4.9.0", "targetTag" : "latest"}'
```

### 7.15.4 | Push and Pull Docker Images

This section includes instructions for pushing and pulling Docker images using a Docker client configuration with Artifactory.

#### 7.15.4.1 | Docker Client Push and Pull Commands

To get the corresponding `docker push` and `docker pull` commands for any repository, go to [Artifactory | Artifacts | Artifact Repository Browser](#), in the **Application** module, and click **Set Me Up**.

Docker

i

i

i

i

Podman

i

#### 7.15.4.2 | Push Multi-Architecture Docker Images to Artifactory

JFrog Artifactory supports the following methods for pushing multi-architecture Docker images to a Docker Registry:

- Push Docker Images One by One
- Push Docker images in bulk using the Docker Buildx CLI (Supported from Artifactory 7.21.2 and higher)
- Push Multi-Architecture Docker Images Using Docker Build from Artifactory 7.21.2 and higher.

#### 7.15.4.3 | Push Docker Images One by One

##### Backward Compatibility

To learn how the standard Docker Pull REST API functions in Artifactory 7.21.2, see Pushing Multi-Architecture Docker Images Using Docker Build.

You can push multi-architecture Docker images, using a 'Manifest Lists' file, (officially referred by Docker as the 'fat manifest file'), which references image manifests for platform-specific versions of an image. For more information, click [here](#).

The process of pushing multi-architecture Docker images is similar to the standard Docker Push process, with a few exceptions:

# JFrog Artifactory Documentation

## Displayed in the header

1. Each architecture gets a different tag.
2. After all the architectures have been built and pushed, a single 'fat manifest file' is created and contains all of the images with the relevant tagging.
3. After pushing the 'fat manifest file', the images are published with the given tags.

```
$ docker build -t domain/docker/multiarch-image:amd64 --build-arg ARCH=amd64/<docker_file>
$ docker push domain/docker/multiarch-image:amd64

$ docker build -t domain_name: port/docker/multiarch-image:arm64 --build-arg ARCH=arm64/<docker_file>
$ docker push domain/docker/multiarch-image:arm64

$ docker manifest create \
domain_name:port1/docker/multiarch-image:tag \
--amend domain/docker/multiarch-image:amd64 \
--amend domain/docker/multiarch-image:arm64 \
$ docker manifest push domain/docker/multiarch-image:my-tag
```

### 7.15.4.4 | Push Images in Bulk Using the Docker Buildx CLI

From Artifactory 7.21.2, the Docker `buildx` command is supported, allowing you to create and upload Docker 'manifest lists' to the Docker registry in Artifactory. Docker `buildx` allows you to build and push multi-architecture images using a single command instead of having to build and push each of the architecture images separately. For more information, see [Working with buildx](#).

To support the Docker `buildx`, Artifactory saves each architecture of the image under the following path structure with the tag that includes the originally published tag, the image operating system, and the image architecture.

`imageName:tag-os-arch`

The following example shows the Docker BuildX API usage.

```
docker buildx build --platform linux/amd64,linux/arm64 --tag domain/docker/multiarch-image:tag --output=type=image,push=true --push .
```

### 7.15.4.5 | Push Multi-Architecture Docker Images Using Docker Build

From Artifactory version 7.21.2 and higher, if you continue to push multi-architecture Docker images using Docker `build`, all your pushed images will be duplicated, and the architecture tag will be automatically added to each image.

In the following example, pushing the following images using Docker Build will result in Artifactory automatically duplicating the images and adding the `linux` tag to each image.

- **List Manifest**

```
docker.artifactory.<domain_name>/test/busybox:1.33
```

- **Image A**

1. Image A is pushed during the build.

```
docker.artifactory.<domain_name>/test/busybox:1.33-amd64
```

2. Artifactory duplicates the image and adds the '`linux`' tag.

```
docker.artifactory.<domain_name>/test/busybox:1.33-linux-amd64
```

- **Image B**

1. Image B is pushed during the build.

```
docker.artifactory.<domain_name>/test/busybox:1.33-s390x
```

2. Artifactory duplicates the image and adds the '`linux`' tag.

```
artifactory.us..<domain_name>/test/busybox:1.33-linux-s390x
```

### 7.15.5 | Browse Docker Repositories

For general information on how to browse repositories, please refer to [Browsing Artifacts](#).

The **Docker Info** tab includes three sections:

- **Tag Info**
- **Docker Tag Visualization**
- **Labels**

#### 7.15.5.1 | Docker Tag Info

Presents basic details about the selected tag.

:

# JFrog Artifactory Documentation

## Displayed in the header

Field	Description
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. <b>Tip</b> Click the label count to view the attached labels at the bottom of the screen.

### 7.15.5.2 | 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
<code>&lt;/&gt;</code>	The layer ID
	The layer size
	The timestamp when the layer was created
<code>&gt;</code>	The command that created the layer
	The digest

### 7.15.5.3 | Docker Labels

This section displays the labels attached to the image.

Note also, that 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.

7.15.5.4 | [List Manifest Content](#)

Starting from Artifactory version 7.81.0, this section displays the manifests nested under a list manifest in the tag folder and the `list.manifest.json` file in the **Artifacts** page. You can click the

icon or the SHA value to focus on the artifact in the Artifacts page, without manually searching for it. Unavailable manifests will show on the table as greyed out.

Field	Description
Manifest	The SHA256 value of the nested image
OS/Arch	The operating system and architecture of the image
Size	The size of the manifest file

#### 7.15.6 | Search for Docker Images

You can search for Docker images by their name, tag or image digest using the Artifact Package Search or through the REST API.

#### 7.15.7 | List Docker Images

Artifactory supports the following REST API endpoints related to Docker registries:

- List Docker Repositories 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.

Artifactory also supports pagination for this endpoint.

To enable fetch from cache using the List Docker Repositories and the List Docker Tags REST APIs, set the `artifactory.docker.catalogs.tags.fallback.fetch.remote.cache` system property to true (default false) in the `artifactory.system.properties` file:

```
## Enable fetch from cache in Docker repositories
#artifactory.docker.catalogs.tags.fallback.fetch.remote.cache=true
```

Artifactory needs to be restarted for this change to take effect.

#### 7.15.8 | Deletion and Cleanup of Docker Tags and Repositories

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>/artifactory/<Docker v2 repository name>/<image namespace>"
```

Or for a specific tag version:

```
curl -u<user:password> -X DELETE "<Artifactory URL>/artifactory/<Docker v2 repository name>/<image namespace>/<tag>"
```

For example, to remove the latest tag of an Ubuntu repository:

```
//Removing the latest tag from the "jfrogs/ubuntu" repository
curl -uadmin:password -X DELETE "https://artprod.company.com/artifactory/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.

#### 7.15.9 | Delete Multi-Architecture Docker Tags

You can delete multi-architecture image tags and all of the architectures under them at once using two ways:

- Via the JFrog Platform WebUI (Supported from version 7.98.2)
- Via curl (Supported from version 7.91.0)

#### Via the JFrog Platform WebUI

Starting from Artifactory version 7.98.2, Artifactory supports deleting image tags containing a `manifest.list` with several architectures at once through the JFrog Platform WebUI.

# JFrog Artifactory Documentation

## Displayed in the header

To delete a multi-architecture tag:

1. Find the tag you would like to delete in the Artifact tree, and right-click the tag name

2. Select **Delete Architectures** from the options menu

3. The window will show a list of the architectures that will be deleted: review the list and when you are sure, click **Delete All**

### Via Curl

From version 7.91.0, Artifactory supports deleting multi-architecture Docker tags with all of their sub-architectures. To delete a multi-architecture tag using curl, execute the following command:

```
curl -H "Authorization: Bearer <TOKEN>" -X DELETE -H "Content-Type: application/json" "https://<JFROG_HOST_URL>/artifactory/api/docker/<REPOSITORY_NAME>/v2/delete" -d '  
{  
    "dockerRepository": "<PATH_TO_IMAGE_NAME>",  
    "tag": "<TAG>"  
}'
```

where:

Parameter	Description
dockerRepository	The docker repository and path to the image for deletion
tag	The docker image tag that you want to delete

### Note

Make sure to replace the placeholders in bold with your own token, host URL, repository name, path to image name, and tag.

For example, the following curl command will delete the Docker image library/ubuntu with all of its sub-architectures from the docker-prod repository:

```
curl -H "Authorization: Bearer EIGU0YXHcuaRdf3piiuJ6t7kWFpIukX691fGXwPlojMtE64siym5S4MLVRdu43gq" -X DELETE -H "Content-Type: application/json" "https://my-awesome.jfrog.io/artifactory/api/docker/docker-prod/v2/delete" -d '  
{  
    "dockerRepository": "library/ubuntu",  
    "tag": "1.0.0"  
}'
```

### 7.15.10 | View Docker Build Information

You may store exhaustive build information in Artifactory by running your Docker builds with JFrog CLI.

JFrog CLI collects build-info from your build agents and then publishes it to Artifactory. Once published, the build info can be viewed in the Build Browser under **Builds**.

For more details on Docker build integration using JFrog CLI, please refer to [Managing Docker Images](#) in the JFrog CLI User Guide.

### 7.15.11 | Tag Retention Logic

Docker, OCI, and Helm OCI local repositories include the ability to define repository-specific retention policies using the Max Unique Tags and Tag Retention fields. These fields allow you to control how Artifactory behaves when pushing new images and tags, and how it deals with tag overwriting.

In this article, learn how each field works and the implications when both settings are active together.

#### Important

Before version 7.75.3, the tag retention logic applied only to the tag entities themselves, but not to their layers or blobs. This meant that when a manifest exceeded the limits and was deleted, Artifactory continued to store its nested images, layers, and blobs, although their reference tag was removed. This led to unnecessary utilization of storage.

"Tag retention logic" refers to two fields that are available in Docker, OCI, and Helm OCI local repository settings when creating or editing a repository:

- Max Unique Tags
- Tag Retention



By default, Max Unique Tags is set to zero (0) and Tag Retention is set to one (1), which means that they are inactive. With these default values, Artifactory allows uploading an unlimited number of unique tags and does not save previous overwrites of existing tags. However, this behavior applies specifically when overriding a manifest file. When overriding a list manifest, there are two options:

- Only the list manifest is overwritten, the associated sub-manifests remain unchanged (default behavior).
- Both the list manifest and associated sub-manifests are overwritten. To use this option, set the system property `artifactory.docker.complete.manifest.override.enabled = true`.

When entering a higher limit value, these fields become active and apply FIFO (first in, first out) logic, meaning that the oldest tag is removed when the number of tags or overwrites exceeds the defined limits.

#### 7.15.11.1 | Use Max Unique Tags

This field allows you to set a numeric limit to how many unique tag names your local repository can hold. This field uses FIFO (first in, first out) logic meaning that once you upload a new unique tag name that exceeds the set limit, your oldest tag will be removed and your new unique tag will be stored instead.

By default this field is set to zero (0), meaning that there is no limit and you may upload as many unique tag names as you like.

#### Example of Max Unique Tags Usage

In your repository, you have three unique image tag names that were pushed to your repository in this order: V 1.0.0, V 1.0.1, and V 2.0.0 (V 1.0.0 being the first, and V 2.0.0 the last).

In this repository, Max Unique Tags is set to 3, meaning that at any given time, you can only store 3 unique tag names: V 1.0.0, V 1.0.1, and V 2.0.0.

#### Note

In the diagrams, the green boxes = stored tags, and the gray boxes = deleted tags.

Next, you upload an additional tag: V 2.0.1. Because V 2.0.1 has a new and unique tag name, it will exceed the Max Unique Tags limit that you have set on your repository, and therefore your oldest tag, which is V 1.0.0, will be removed to make space for the new V 2.0.1.

The result would show that your repository now stores V 1.0.1, V 2.0.0, and V 2.0.1.

#### Note

If your artifactory instance has Trash Can enabled, you will be able to retrieve removed tags if needed.

##### 7.15.11.2 | Use Tag Retention

The Tag Retention field controls how many overwrites of the same tag are saved in Artifactory. Tag overwriting occurs when you upload a new revision of a tag name that already exists in your repository: for instance, "latest".

#### Note

The Tag Retention limit applies to any tag name you overwrite, not just "latest".

By default, this field is set to one (1), implying that if you upload a tag, Artifactory will not save the previous version of the tag that is being overwritten.

When setting this field to a value of 2 (or above), Artifactory will automatically make a copy of the existing tag in Artifactory before it is overwritten and rename it according to its SHA2 hash value. This field accepts any number that represents a limit of how many previous revisions of the same tag should be stored. This field also works according to FIFO (first in, first out) logic, meaning that once you overwrite your tag beyond the limit, the oldest tag revision will be removed.

#### Note

If your artifactory instance has Trash Can enabled you will be able to retrieve your oldest overwritten tag according to its SHA2 hash value if needed.

#### Example of Tag Retention Usage

The following diagram shows what happens when you overwrite the same "Latest" tag over and over again three times:

- 1st push: Latest (Original)
- 2nd push: Latest (revision 2)
- 3rd push: Latest (revision 3)

#### Note

In the diagrams, the green boxes = stored tags, and the gray boxes = deleted tags.

During this process, once we upload the Latest tag for the 2nd time, the original copy of Latest already stored in Artifactory is renamed according to its hash and continues to be stored in the system, making way for the updated revision of Latest that will use the "Latest" tag name. Then, when the 3rd revision of Tag1 is pushed, the 2nd revision of the second version is renamed as a SHA value.

Finally, when uploading the Latest tag for the 4th time, the number of overwritten tags exceeded the Tag Retention limit of 3, meaning that the oldest iteration of Latest tag (SHA 1), the original first push, will be removed.

#### 7.15.11.3 | Using Both Max Unique Tags and Tag Retention

The main implication of applying both Max Unique Tags and Tag Retention together is that Max unique tags takes precedence and will remove all retention revisions associated with the current tag if it is being deleted by the logic. This means that if a tag is removed due to exceeding the Max Unique Tags limit (being the oldest unique tag when a new one has been pushed), all of its Tag Retention copies will be removed as well.

This prevents Artifactory from storing redundant revision copies of unneeded tags if they are removed by the Max Unique Tags logic. Therefore, Tag Retention duplicates continue to aggregate with each tag overwrite, as long as the current tag remains within the Max Unique Tags limit.

This interaction between Max unique tags and Tag retention is best understood as a matrix. Looking at the following diagram, the X axis (left to right) represents unique tags being pushed into your repository. The Y axis (top to bottom) represents overwrites of the same tag over and over again.

**Note**

In the diagrams, the green boxes = stored tags, and the gray boxes = deleted tags.

In this example, we have set the following limits:

- Max Unique Tags = 3
- Tag Retention = 3

These limits are represented as a blue area applied to both axes.

When you exceed the Max Unique Tags limit, the oldest unique tag will be removed from Artifactory. Since the unique tag is considered a lead tag, all of its overwrite revisions that are associated with it will also be removed. In the following example, the moment tag V 2.0.1 is uploaded, then V 1.0.0 is removed (since it exceeded the max unique tags settings), and together with it, all SHA1, SHA2, and SHA3 are previous revisions of V 1.0.0.

# JFrog Artifactory Documentation Displayed in the header

## 7.15.11.4 | How Tag Retention Logic Affects Manifest Lists

The Docker/ OCI upload APIs have been enhanced to add relationship markers between manifests and manifest lists. This allows us to apply the tag retention logic of both Max Unique Tags and Tag Retention, to remove a manifest list including all of its referenced sub-manifests as a single complete tag context.

If we have a `list.manifest` with several manifests under it, the tag retention logic remains the same and simply renames that tag with all its sub-manifests as a whole. Manifests that are nested under the list are treated according to their relationship mapping and removed when they exceed the limits.

## Overwriting List Manifests with a Manifest and Vice-Versa

The base principle in Artifactory is that if you have decided to rewrite a tag, you are performing a complete replacement. Therefore if you decide to overwrite a `list.manifest` with a single manifest, all of the sub-manifests related to the original revision will also be removed. When tag retention is applied, the complete structure of the previous revision is saved including both the `list.manifest` and all its sub-manifests all together.

## 7.15.11.5 | Limitations of Tag Retention Logic

- Multi-architecture retention logic is not backward compatible: the new relationship logic between `list.manifests` and sub-manifests will only apply to tags uploaded via the updated Docker/ OCI upload service. All previously stored tags and images will not be removed.

To apply and maintain this relationship logic, Artifactory will mark items during the upload process so it can track the tags during the lifecycle. Older tags would have no markers, and therefore Artifactory would not be able to identify the relationship and remove items accordingly.

- Removed tags go to the [Trash Can](#) and can be restored, but not all JFrog environments have the Trash Can enabled. Before setting tag retention logic, ensure that your Trash Can is enabled.
- Max unique tags logic currently does not exclude the Latest tag. If for some reason you upload several unique tags that do not include Latest, your latest tag in artifactory will be removed by the FIFO logic just like any other tag.
- The Max unique tags logic uses lazy invocation, meaning that it is triggered per package push command. Once you apply the limit, the removal of exceeding tags will only occur on the specific package you have pushed to Artifactory. Currently, there is no process that runs on all your packages.

## 7.15.12 | Working with Docker Content Trust

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.

### Note

Artifactory supports hosting signed images without the need for any additional configuration.

#### 7.15.12.1 | Configure 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 Docker Notary Hosts File
- Configure the Docker Notary Server
- Configure the Docker Client with Docker Content Trust
- Test Your Docker Content Trust Setup

#### 7.15.12.1.1 | Configure Your Docker Notary 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'
```

#### 7.15.12.1.2 | Configure the Docker 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 ubuntu

RUN apt-get update \
&& apt-get install -y \
tree \
vim \
git \
ca-certificates \
curl \
--no-install-recommends
RUN install -m 0755 -d /etc/apt/keyrings \
&& curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc \
&& chmod a+r /etc/apt/keyrings/docker.asc

RUN echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
tee /etc/apt/sources.list.d/docker.list > /dev/null

RUN apt-get update && apt-get -y install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

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/ca-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 Docker file.

**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
```

#### 7.15.12.1.3 | Configure the Docker Client with Docker Content Trust

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
```

### 7.15.12.2 | Test Your Docker Content Trust 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:

- Artifactory is up and running in a Docker container
- You have configured the Notary server
- 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'
```

#### Pull an image for testing

```
docker pull docker/trusttest
```

After you have pulled the image, you need to docker login to 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]

The Docker image is signed after it is pushed to Artifactory.

### 7.15.13 | Docker Advanced Topics

This page contains advanced configuration options for Docker with Artifactory, and includes the following sections:

- Use a Self-Signed SSL Certificate with Docker
- Use Your Own Certificate with Nginx
- Set Your Docker Credentials Manually
- Authenticate Docker Via OAuth

#### 7.15.13.1 | Use a Self-signed SSL Certificate with Docker

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 see 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**, 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
```

#### 7.15.13.2 | Use Your Own Certificate with Nginx

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.

# JFrog Artifactory Documentation

## Displayed in the header

For details, see Set TLS on the JFrog Platform.

### 7.15.13.3 | Set Your Docker Credentials Manually

If you are unable to log in to Docker, you may need to set your credentials manually.

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`

#### Get `.dockercfg` entries directly from Artifactory

```
$ curl -uadmin:password "https://artprod.company.com/<v1|v2>/auth"
{
  "https://artprod.company.com" : {
    "auth" : "YWRtaW46QVAlN050aHZTMnM5Qk02RkR5RjNBVmF4TVF1",
    "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"
  }
}
```

### 7.15.13.4 | Authenticate Docker via OAuth

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 Users** is selected to make sure a user record is created for you the 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.

### 7.15.14 | Get Started With Artifactory as a Docker Registry

There are these main ways to get started using Docker with Artifactory:

- Get Started with Docker and Artifactory Cloud
- Get Started with Docker and Artifactory Self-Hosted

#### 7.15.14.1 | Get Started with Docker and Artifactory Cloud

Using Docker repositories with Artifactory Cloud is quick and easy. With Artifactory Cloud, you are using Artifactory as a hosted service and 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.

##### 7.15.14.1.1 | Use Kubernetes with Artifactory Cloud

Follow this guide to configure your Kubernetes server with an Artifactory container registry, and be able to pull your images from a private Artifactory registry.

To integrate Artifactory with Kubernetes, you need:

- An artifactory instance with a configured Docker repository: for more information, see [Set Up a Docker Repository](#).
- A Kubernetes cluster.

To configure Kubernetes to pull containers from a private Artifactory registry:

# JFrog Artifactory Documentation

## Displayed in the header

- For each relevant namespace, create a Kubernetes docker-registry secret for connecting to your Artifactory by running the following command:

```
kubectl create secret docker-registry regcred \
--docker-server=<JFROG-HOSTNAME> \
--docker-username=<JFROG-USERNAME> \
--docker-password=<PASSWORD> \
--docker-email=<EMAIL> \
--namespace <NAMESPACE>
```

### Note

Make sure to replace the placeholders with your actual user information: for security reasons, it is best to choose a dedicated user which is not your Artifactory admin and has minimal required permissions.

Variable	Description
<b>JFROG-HOSTNAME</b>	Your JFrog hostname URL
<b>JFROG-USERNAME</b>	Your JFrog account username
<b>PASSWORD</b>	Your secret or JFrog identity token. Note that you can edit the token scope to restrict access to Artifactory.
<b>EMAIL</b>	The email address associated with your JFrog account
<b>NAMESPACE</b>	Your Kubernetes cluster namespace

For example:

```
→ ~ kubectl create secret docker-registry regcred \
--docker-server=my-artifactory.jfrog.io \
--docker-username=read-only \
--docker-password=my-super-secret-pass \
--docker-email=johndoe@example.com \
--namespace my-app-ns
```

- Set up Kubernetes to use the secret to pull images for your workloads. You can do this either for all the workloads in your namespace, or for each workload separately.

- To set the secret as default to your namespace (recommended), run the following command to edit your Service Account object and add your secret name into the `imagePullSecrets` list attribute:

### Note

Make sure to replace the placeholder with your actual Kubernetes namespace

```
→ ~ kubectl edit serviceaccount default -n <NAMESPACE>
apiVersion: v1
kind: ServiceAccount
imagePullSecrets:
- name: regcred
...
```

- To add the secret to every workload separately, add it into your object manifests and helm charts, see the below example:

```
apiVersion: apps/v1
kind: Deployment
...
spec:
...
template:
  spec:
    containers:
      - image: my-artifactory.jfrog.io/default-docker-virtual/my-app:1.0.1
        imagePullSecrets:
          - name: regcred
```

- Test your configuration by running the following command:

### Note

Make sure to replace the placeholder with your actual Kubernetes namespace

```
→ ~ kubectl get pods -n <NAMESPACE>
```

You should get the following response:

```
NAME           READY   STATUS    RESTARTS   AGE
my-app-57db67b7d5-nr8db  1/1     Running   0          5m
```

## Amazon EKS Integration

If you are using Amazon EKS, you can use JFrog's seamless integration with AWS AssumeRole which allows JFrog Artifactory to securely serve container images to EKS. For more information, see [Empowering Kubernetes Security: JFrog's Seamless Integration with AWS AssumeRole](#).

# JFrog Artifactory Documentation

## Displayed in the header

For more information about Artifactory and Kubernetes, see [Kubernetes Helm Chart Repositories](#).

### 7.15.14.1.2 | Use Docker Client with Artifactory Cloud

To use the Docker client with one of your Artifactory Cloud Docker repositories, you can use the native Docker client to login to each Docker repository, pull, and push images.

- Log in to your repository use the following command with your Artifactory Cloud credentials.

```
docker login ${server-name}.jfrog.io
```

- Pull an image using the following command.

```
docker pull ${server-name}.jfrog.io/{repo-name}/<image name>
```

- Push an image by first tagging it and then using the push command.

```
docker tag <image name> ${server-name}.jfrog.io/{repo-name}/<image name>
docker push ${server-name}.jfrog.io/{repo-name}/<image name>
```

### 7.15.14.1.3 | Test Your Docker and Artifactory Cloud Setup

The following example demonstrates the following scenario:

- Pulling the hello-world Docker image
- Logging into your virtual Docker repository
- Retagging the hello-world image, and then pushing it into your virtual Docker repository

#### Note

In this example, the Artifactory Cloud server is named **acme**.

1. Start by creating a virtual Docker repository called dockerv2-virtual.

2. Pull the hello-world image

```
docker pull hello-world
```

3. Log in to repository dockerv2-virtual

```
docker login acme-dockerv2-virtual.jfrog.io
```

4. Tag the hello-world image

```
docker tag hello-world acme-dockerv2-virtual.jfrog.io/hello-world
```

5. Push the tagged hello-world image to dockerv2-virtual

```
docker push acme-dockerv2-virtual.jfrog.io/hello-world
```

### 7.15.14.2 | Get Started with Docker and Artifactory Self-Hosted

The Docker client has the following two limitations:

1. You cannot use a context path when providing the registry path (e.g localhost:8082/artifactory is not valid)
2. Docker will only send basic HTTP authentication when working against an HTTPS host or when using the insecure registry flag

Artifactory offers solutions to these limitations allowing you to create and use any number of Docker registries.

- **Using a reverse proxy** When used, a reverse proxy, maps Docker commands to one of the multiple Docker registries in Artifactory
- **Without a reverse proxy** From version 5.8, Artifactory supports using Docker without the use of a reverse proxy allowing you to create and use multiple Docker registries in Artifactory out-of-the-box.

### 7.15.14.2.1 | Get Started with Docker Using a Reverse Proxy

When using Artifactory with a reverse proxy, you need to map Docker commands to Docker registries in Artifactory using either the subdomain method, ports method or repository path 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.

The ports method maps a port number 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.

The repository path method allows a single point of entry (URL) to access different repositories. This is done by embedding the name of the repository being accessed into the image path.

If a wildcard certificate is available, we recommend the subdomain method since it will only require a one-time effort and follows the Docker convention more closely.

7.15.14.2.1.1 | [The Subdomain Method for Docker](#)

Getting started with Docker and your self-hosted Artifactory Pro installation using the subdomain method involves four basic steps:

1. Configure Artifactory
2. Configure your reverse proxy
3. Configure your Docker client
4. Testing your setup

### Configure Artifactory

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 virtual Docker repository (as well as a local and remote Docker repository that it should aggregate). In our example below we will use a repository named **docker-virtual**.
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. To create a self-signed certificate, you can follow these instructions for Ubuntu.

#### Note

Make sure your certificate matches the **Artifactory hostname** used in your reverse proxy configuration. In our example below we will use **art.local**.

### Configure Your Reverse Proxy

Artifactory's can generate your complete reverse proxy configuration file for supported servers.

Go to **Reverse Proxy Configuration Generator** and fill in the fields according to how your reverse proxy is set up while making sure to:

1. Use the correct **Artifactory hostname** in the Public Server Name field of **Reverse Proxy Settings** (in our example this will be **art.local**)
2. Select **Subdomain** as the **Reverse Proxy Method** under Docker Reverse Proxy Settings.

7.15.14.2.1.2 | [Nginx for Docker](#)

Copy the code snippet generated by the **configuration generator** into your **artifactory-nginx.conf** file, and place it in your **/etc/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
```

7.15.14.2.1.3 | [Apache HTTPD for Docker](#)

Copy the code snippet generated by the **configuration generator** into your **artifactory-apache.conf** file and place it in your **/etc/apache2/sites-available** directory.

# JFrog Artifactory Documentation

## Displayed in the header

Create the following symbolic link:

```
sudo ln -s /etc/apache2/sites-available/artifactory-apache.conf /etc/apache2/sites-enabled/artifactory-apache.conf
```

### Note

The `httpd.conf` header file should be tuned to work correctly using these values to avoid errors:

```
# Apache libraries location (should be tuned)
Define APACHE_LIB_DIR /usr/lib/apache2/modules      <---- Input field in UI could be great

# Apache Logs (default : beside other Artifactory logs)
Define APACHE_LOG_DIR /[TheArtifactoryHome]/logs      <---- ErrorLog/CustomLog could be uncommented & Input field in UI could be great

LoadModule proxy_module ${APACHE_LIB_DIR}/mod_proxy.so
LoadModule rewrite_module ${APACHE_LIB_DIR}/mod_rewrite.so
LoadModule proxy_ajp_module ${APACHE_LIB_DIR}/mod_proxy_ajp.so
LoadModule proxy_http_module ${APACHE_LIB_DIR}/mod_proxy_http.so
LoadModule ssl_module ${APACHE_LIB_DIR}/mod_ssl.so      <---- Only if HTTPS used

Listen 443      <---- Only if HTTPS used
Listen XXX      <--- All port used as Docker registry
```

7.15.14.2.1.4 | [Test Your Docker Reverse Proxy 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>/api/system/ping
```

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-virtual  
`docker login docker-virtual.art.local`
- Tag the "hello-world" image  
`docker tag hello-world docker-virtual.art.local/hello-world`
- Push the tagged "hello-world" image to docker-virtual  
`docker push docker-virtual.art.local/hello-world`

7.15.14.2.1.5 | [The Repository Path Method for Docker](#)

Getting started with Docker and your self-hosted Artifactory Pro installation using the path method involves four basic steps:

1. Configuring Artifactory
2. Configure your Reverse Proxy
3. Configure your Docker client
4. Test your setup

### Configuring Artifactory

To configure Artifactory and your reverse proxy using the path method, carry out the following steps:

1. Make sure Artifactory is up and running, and is activated with a valid license.
2. Create your **Virtual Docker Repositories** (as well as a local and remote Docker repository that it should aggregate). In our example below we will use a repository named **docker-virtual**.
3. Make sure you have a reverse proxy server up and running.
4. Obtain a valid SSL certificate or use a self-signed certificate. To create a self-signed certificate, you can follow [these instructions for Ubuntu](#).

### Note

Make sure your certificate matches the **Artifactory hostname** used in your reverse proxy configuration. In our example below we will use **art.local**.

### Configure your reverse proxy

Artifactory's can generate your complete reverse proxy configuration file for supported servers.

Go to **Reverse Proxy Configuration Generator** and fill in the fields in according to how your reverse proxy is set up while making sure to:

1. Use the correct **Artifactory hostname** in the Reverse Proxy Settings window, configure the **Public Server Name** field (in our example this will be `art.local`)
2. Select **Repository Path** 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/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

# JFrog Artifactory Documentation

## Displayed in the header

Copy the code snippet generated by the configuration generator into your artifactory-apache.conf file and place it in your /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
```

### Configure 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 as described in the Docker documentation.

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>/api/system/ping
```

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-virtual

```
docker login art.local
```

- Tag the "hello-world" image

```
docker tag hello-world art.local/docker-virtual/hello-world
```

- Push the tagged "hello-world" image to docker-virtual

```
docker push art.local/docker-virtual/hello-world
```

### Configure Artifactory with Repository Path Method

To configure Artifactory to use the Repository Path method, carry out the following steps:

1. Make sure Artifactory is up and running, and is activated with a valid license.
2. Create your **virtual Docker repository** (as well as a local and remote Docker repository that it should aggregate). In our example below we will use a repository named **docker-virtual**.
3. Go to the **HTTP Settings** page from the **Administration** module under **Artifactory | General | HTTP Settings**. In the **Docker Settings** panel, select **Repository Path** as the Docker Access Method.

In the **Reverse Proxy Settings** panel select **Embedded Tomcat** as the Server Provider (which indicates you're not using a reverse proxy).

### You must use Embedded Tomcat

You can only use Artifactory as a Docker registry without a reverse proxy by using the internal embedded Tomcat.

7.15.14.2.1.6 | [The Ports Method for Docker](#)

Getting started with Docker and your self-hosted Artifactory Pro installation using the ports method involves two basic steps:

1. Configure Artifactory and your reverse proxy
2. Configure your Docker client

Configure 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 Virtual Docker Registry (as well as a local and remote Docker repository that it should aggregate). In our example below we will use a repository named **docker-virtual**.
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.

#### Note

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:

1. Use the correct **Artifactory hostname** in the Public Server Name field
2. 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-virtual`.

#### 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:

# JFrog Artifactory Documentation

## Displayed in the header

- 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
```

# JFrog Artifactory Documentation Displayed in the header

```
# 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 198.168.1.206:8082
#
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>
```

## Configure 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>/api/system/ping
```

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-virtual repository that is bound to port 5001:

```
// Pull the "hello-world" image
docker pull hello-world

// Login to repository docker-virtual
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-virtual
docker push art-local:5001/hello-world
```

## Test 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 as described in the Docker documentation.
2. 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).  
Running `$docker info` will list the Insecure registries that have been applied under the Insecure Registries entry.
3. Use the steps above to interact with the Artifactory Docker Registry

## 7.15.14.2.2 | Get Started with Docker Without a Reverse Proxy

Previously, Artifactory supported the Ports and Subdomain methods described above when using a reverse proxy. From **version 5.8**, Artifactory introduces a new method referred to as the "Repository Path" method since it uses the Docker repository path prefix (<REPOSITORY\_KEY/IMAGE>) to access a specific Artifactory Docker registry from the Docker client. Note that you may still have a reverse proxy configured for Artifactory for other reasons, however, when configured to use Repository Path method, requests to Docker registries in Artifactory will be handled by Artifactory's embedded Tomcat instead of the reverse proxy.

### Docker API v2 required

You can only use the Repository Path method with Artifactory Docker registries configured for Docker API v2.

7.15.14.2.2.1 | [Configure Your Docker Client Without Reverse Proxy](#)

Using the Repository Path method, you can work with Artifactory as a Docker registry without a reverse proxy on an insecure connection (i.e. only HTTP is supported, not HTTPS). **You need to configure the Docker client to work with an insecure registry** as described in the [Docker documentation](#).

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). Running `$docker info` will list the Insecure registries that have been applied under the Insecure Registries entry.

7.15.14.2.2.2 | [Test Your Docker Without Reverse Proxy Setup](#)

### Don't use localhost or 127.0.0.1 or "/artifactory"

Due to a limitation in the Docker client, you cannot access an Artifactory Docker registry as localhost or 127.0.0.1. If you need to access a local installation of Artifactory, make sure to specify its full IP address.

In addition, when specifying Artifactory's URL, you should omit the /artifactory suffix normally used.

For example, if your local machine's IP address is 10.1.16.114, then you must specify your Artifactory URL as `http://10.1.16.114:8082` (using `http://localhost:8082` will not work).

The code snippets below assume you have a virtual Docker repository named `docker-virtual` in an Artifactory installation at IP 10.1.16.114.

First, you should verify that your Docker client can access Artifactory by run the following command. Making sure that the return code is 200:

```
curl -I -k -v http://10.1.16.114:8082/artifactory/api/system/ping
```

Now you can proceed to test your Docker registry.

- Login to Artifactory as your Docker registry  
`docker login -u admin -p password 10.1.16.114:8082`
- Pull the hello-world image from the docker-virtual repository  
`docker pull 10.1.16.114:8082/docker-virtual/hello-world:latest`
- Tag a Docker image  
`docker tag 10.1.16.114:8082/docker-virtual/hello-world:latest 10.1.16.114:8082/docker-virtual/<tag_name>`
- Push the tagged image to docker-virtual  
`docker push 10.1.16.114:8082/docker-virtual/<tag_name>`

### 7.15.15 | Migrate 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 [Migrate a V1 repository to V2](#) under the [Use 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 [Use Docker V1](#).

### 7.15.16 | Use Docker V1

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 [Get Started with Artifactory as a Docker Registry](#).

#### Note

Docker V1 is not supported on Cloud instances.

#### 7.15.16.1 | Get Started with Artifactory and Docker V1

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. Set up NGINX as a Reverse Proxy for Docker V1

Artifactory can only be used with Docker through a reverse proxy due to the following limitations of the Docker client:

# JFrog Artifactory Documentation Displayed in the header

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/docker/docker-local/v1;  
        }  
    }  
}
```

## Multiple Docker repositories and port bindings

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. Create a Local Docker V1 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.



## Work with Artifactory Cloud For Docker V1

Due to limitations of the Docker client, in Artifactory Cloud 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. Set Up Authentication for Docker V1

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. Push and Pull Images With Docker V1

Pushing and pulling images when using Docker V1 is done in the same way as when using Docker V2.

7.15.16.1.1 | [Watch the Docker V1 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.



7.15.16.2 | [Browse Docker V1 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.imageId:** 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

#### View 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.

## View 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.

## Search 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`).

### Promote Docker Images

Promoting Docker images with Docker V1 is done in exactly the same way as when Promoting Images with Docker V2.

#### 7.15.16.3 | Migrate a Docker 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>"
}
```

# JFrog Artifactory Documentation

## Displayed in the header

where:

Parameter	Description
<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 <code>migrate</code> 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 its 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"}'
```

### 7.15.16.4 | Docker V1 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).

### 7.15.16.5 | Docker V1 Advanced Topics

This topic discusses advanced configuration options for Docker V1 and contains the following sections:

- Use a Self-Signed SSL Certificate for Docker V1
- Docker V1 Alternative Proxy Servers
- Port Bindings for Docker V1
- Docker V1 Repository Path and Domain

#### 7.15.16.5.1 | Use a Self-signed SSL Certificate for Docker V1

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 ).

# JFrog Artifactory Documentation Displayed in the header

7.15.16.5.2 | Docker V1 Alternative Proxy Servers

In addition to NGINX, you can setup Artifactory to work with Docker using Apache.

The sample configuration below configures SSL on port 443 and a server name of artprod.company.com.

```
<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>
```

7.15.16.5.3 | Port Bindings for Docker V1

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.

7.15.16.5.4 | Docker V1 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.

## 7.16 | Git LFS Repositories

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.

**Integration Benefits JFrog Artifactory and Git Repository**

### 7.16.1 | Set Up a Git LFS Repository

This section contains the following topics

- Set Up Local Git LFS repositories
- Set Up Remote Git LFS repositories
- Set Up Virtual Git LFS repositories
- Set Up the Git LFS Client to Point to Artifactory

### 7.16.1.1 | Set Up Local Git LFS Repositories

To create a Git LFS local repository and enable calculation of LFS package metadata, from the **Administration** module, select **Repositories | Repositories | Local** and set **Git LFS** as the **Package Type**.

### 7.16.1.2 | Set Up Remote Git LFS Repositories

You can create a Git LFS **remote repository** to proxy Git LFS local repositories on other Artifactory instances and enjoy all the features of a **Smart Remote Repositories**.

To define a Git LFS remote repository, from the **Administration** module, select **Repositories | Repositories | Remote** and set its Package Type to be **Git LFS**, and set the URL of the repository you want to proxy.

### 7.16.1.3 | Set Up Virtual Git LFS 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, from the **Administration** module, select **Repositories | Repositories | Virtual** and 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.

#### 7.16.1.4 | Set 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 Git LFS 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.

## 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 Cloud the URL would be:

```
https://<server name>.jfrog.io/artifactory/api/lfs/<repository key>
```

When configuring replication, reference the repository's browsable url i.e.:

```
http://localhost:8081/artifactory//<repository key>
```

### 7.16.2 | Work with Artifactory and Git LFS without Anonymous Access

By default, Artifactory disallows anonymous access to Git LFS repositories. This is defined in the **Administration** module under **Security | Settings**. For details please refer to Allow Anonymous Access.

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.

### 7.16.3 | Authenticate Git LFS with SSH

Artifactory supports authenticating your Git LFS client via SSH for Self Hosted instances.

#### Not Supported for JFrog Cloud

Git LFS with SSH Authentication is not supported for JFrog Cloud customers.

To authenticate yourself via SSH when using the Git LFS client, execute the following steps:

1. Make sure Artifactory is properly configured for SSH as described in the [SSH Server Configuration](#).

2. Upload your SSH Public Key in the SSH section of your user profile as described in Configuring User Authentication.

3. 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...PC0GuTJ9TlaYD 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://john@myartifactory.company.com:1339/artifactory/lfs-local"
```

#### Note

While the `$USERNAME` field is not mandatory, it is good practice to add it for self-reference.

#### 7.16.4 | Use Git LFS 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.

#### 7.16.5 | Use Git LFS 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/ad1b8d6e1cafdf33e941a5de462ca7edfa8818a70c79feaf68e5ed53dec414c4`

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.

#### 7.16.6 | Git LFS Quick Start Guide

Visit our Knowledge Base for a [quick start guide of Git LFS with Artifactory](#).

### 7.17 | Go Registry

JFrog Artifactory offers support for Go packages, providing:

- Secure, private Go registries with fine-grained access control over Go packages according to projects or development teams.
- Remote Go registries which provide proxy and caching functionality for remote Go resources.
- Enterprise features such as high availability, repository replication for multi-site development and different options for massively scalability storage.
- A full solution with JFrog CLI, allowing you to resolve and publish your Go projects.

#### 7.17.1 | Install the Go Client

Artifactory requires **Go client version 1.11.0** and above.

To install the Go client, please refer to the [Go Programming Language documentation](#).

### Using Homebrew?

You can also install the Go client by running:

```
brew install go
```

### Learn More

[Go CheatSheet for applications using Go](#)

[Integration Benefits JFrog Artifactory and Go Registry](#)

[How to set up a Private, Remote and Virtual Go Registry](#)

### 7.17.2 | Set Up a Go Repository

You can set up the following repository types:

- Local Go Repositories
- Remote Go Repositories
- Virtual Go Repositories

#### 7.17.2.1 | Set Up Local Go Repositories

To deploy Go packages to a local Go registry and enable calculation of Go package metadata, select **Repositories | Repositories | Local** and set **Go** to be the **Package Type** when you create your local repository.

#### 7.17.2.2 | Set Up Remote Go Repositories

### Important

To resolve dependencies from a Remote Go repository, you must nest the remote repository under a virtual Go repository.

A remote Go repository in Artifactory serves as a caching proxy for [golang.org](#), [GitHub.com](#) or a Go repository in a different Artifactory instance.

Artifacts (such as zip files) requested from a remote Go registry 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.

#### 7.17.2.2.1 | Set Up Go Repositories to Proxy Private Registries

Starting from Artifactory version 7.77, It is possible to use remote Go repositories to proxy your private registries hosted on platforms such as:

- GitHub (Cloud)
- GitHub Enterprise (Self-Hosted)
- Bitbucket Cloud (Cloud)
- Bitbucket Server (Self-Hosted)
- GitLab (Cloud & Self-Hosted)

### Important

Before using a Go repository to proxy a private registry, ensure your repository follows the [Go Modules Reference](#), and especially the [Module release and versioning workflow](#).

#### 7.17.2.2.1.1 | Set Up Go Proxy Mirror Repositories

When configuring Artifactory to proxy a private registry, it is highly recommended to always create a proxy mirror directing to <http://proxy.golang.org/> to resolve your public modules. This will reduce your unnecessary usage on your other remotes pointing to your private repositories hosted on other git providers and avoid potential rate-limiting issues.

To create a proxy repository:

1. Create a new remote repository and set **Go** as its **Package Type**.
2. Set the **Repository Key** value, and enter <http://proxy.golang.org/> in the **URL** field.
3. In the **Go Settings** section, set the **Git Provider** as Artifactory.
4. In the **Advanced** tab, select the **Priority Resolution** checkbox.
5. Click **Save & Finish**.

# JFrog Artifactory Documentation

## Displayed in the header

6. Nest the repository you just created and your remote repository proxying a private registry under the same virtual repository. That way, whenever you resolve an artifact through the virtual repository, Artifactory will attempt to resolve your cached artifacts first and avoid requesting the external registry.

7.17.2.2.1.2 | Proxy GitHub with Go

### Go Proxy Best Practice: Add Mirror Repository

We highly recommend that you create a remote repository pointing to the default go mirror ([gocenter.io](https://gocenter.io)) and enter **Artifactory** as the **Git provider**. This way, Artifactory will be able to resolve cached public modules without querying the external registry, reducing the chance of rate limiting. For more information, see [Set Up Go Proxy Mirror Repositories](#).

To create a Remote Repository which serves as a caching proxy for [github.com](https://github.com), follow the steps below:

1. Create a new remote repository and set **Go** to be its **Package Type**.
2. Set the **Repository Key** value, and enter <https://github.com> in the **URL** field as displayed below.
3. In the **Go Settings** set the **Git Provider** as GitHub.

#### Note

When using a remote repository that implements Go API Specification, use Artifactory as the Git provider. For example, to set up a remote registry opposite [proxy.golang.org](https://proxy.golang.org), use Artifactory as the Git provider.

4. In the **Basic** tab, enter the username and password of your GitHub account. This is required due to a rate limit imposed by GitHub when access is anonymous.

#### From November 2020, GitHub no longer accepts account passwords when authenticating via the REST API

As per the latest GitHub update, basic authentication is no longer supported via REST API. Your password must be replaced with a GitHub private access token.

5. In the **Advanced** tab, check **Lenient Host Authentication**.
6. Click **Save & Finish**.

7.17.2.2.1.3 | Proxy GitHub Enterprise with Go

### Go Proxy Best Practice: Add Mirror Repository

We highly recommend that you create a remote repository pointing to the default go mirror ([gocenter.io](https://gocenter.io)) and enter **Artifactory** as the **Git provider**. This way, Artifactory will be able to resolve cached public modules without querying the external registry, reducing the chance of rate limiting. For more information, see [Set Up Go Proxy Mirror Repositories](#).

To create a remote repository that serves as a caching proxy for GitHub Enterprise, follow these steps:

1. Create a new remote repository and set **Go** as the **Package Type**.
2. Set the **Repository Key** value, and enter your local GitHub Enterprise URL in the **URL** field.
3. Enter the user name and personal access token that provides authentication to GitHub Enterprise. For more information about creating a personal access token, see [Generate a Personal Access Token in GitHub Enterprise](#).

4. In the **Git Provider** field, select **GitHub Server** as the Git provider.

5. Click **Save & Finish**.

Generate a Personal Access Token in GitHub Enterprise

This topic describes how to generate a personal access token in GitHub Enterprise that is needed for authentication when using a Go remote repository to proxy your private GitHub Enterprise registry.

**To generate a personal access token in GitHub Enterprise:**

1. Log in to your GitHub Enterprise account.
2. Click your profile photo in the upper-right corner and select **User settings** from the menu.
3. From the sidebar menu on the left, click **Developer settings**.
4. Select **Personal access tokens > Tokens (classic)**.

5. Click **Generate new token**.

6. [optional] In the **Note** field, add a description.

7. In the **Expiration** field, configure when the personal access token will expire [recommended].

8. In the **Select scopes** field, select the **repo** checkbox to assign this set of permissions to the personal access token.

9. [optional] Select additional permissions, if required by your environment.

10. Click **Generate Token**.

7.17.2.2.1.4 | Proxy GitLab with Go

#### Go Proxy Best Practice: Add Mirror Repository

We highly recommend that you create a remote repository pointing to the default go mirror ([gocenter.io](https://gocenter.io)) and enter **Artifactory** as the **Git provider**. This way, Artifactory will be able to resolve cached public modules without querying the external registry, reducing the chance of rate limiting. For more information, see [Set Up Go Proxy Mirror Repositories](#).

To create a remote repository that serves as a caching proxy for GitLab, follow these steps:

1. Create a new remote repository and set **Go** as the **Package Type**.
2. Set the **Repository Key** value, and enter the link to your GitLab environment in the **URL** field:
  - For GitLab Self-managed, enter your environment link
  - For GitLab SaaS, enter <https://gitlab.com/>
3. In the **Go Settings** tab, set the **Git Provider** as **GitLab**.
4. In the **Advanced** tab, enter your GitLab username and password. To generate a GitLab password:
  - a. Go to your GitLab account, click your avatar in the top right corner, and select **Settings**.
  - b. Go to **Access Tokens**, and click **Add new token**.

c. Enter a name and select permissions for your token: select the **api** and **read\_api** permissions, and add other permissions according to your environment. When you are done, click [Create personal access token](#).

d. Copy the token you created, and paste it into the **Password** field in the JFrog platform.

5. Click **Save & Finish**.

6. Create a new virtual repository and under **Repositories**, select the remote repository you have created.

7. Under **Include/ Exclude Patterns**, add the include pattern for GitLab, and click the **+** sign to add the pattern.

- For GitLab Self-managed, enter your environment URL in this structure: `*/<YOUR_URL>/**`
- For GitLab SaaS, enter `*/gitlab.com/**`.

You can use the virtual repository to resolve artifacts from your GitLab private registry.

7.17.2.2.1.5 | Proxy Bitbucket Cloud with Go

#### Go Proxy Best Practice: Add Mirror Repository

We highly recommend that you create a remote repository pointing to the default go mirror ([gocenter.io](https://gocenter.io)) and enter **Artifactory** as the **Git provider**. This way, Artifactory will be able to resolve cached public modules without querying the external registry, reducing the chance of rate limiting. For more information, see [Set Up Go Proxy Mirror Repositories](#).

To create a remote repository that serves as a caching proxy for Bitbucket Cloud, follow these steps:

1. Create a new remote repository and set **Go** as the **Package Type**.
2. Set the **Repository Key** value, and enter <https://bitbucket.org/> in the **URL** field.
3. In the **Go Settings** tab, set the **Git Provider** as **Bitbucket Cloud**.
4. In the **Advanced** tab, enter your Bitbucket username and password. To generate a Bitbucket password:
  - a. Go to your Bitbucket account, click the gear icon in the top right corner, and select **Personal Bitbucket settings**.

b. Go to **App passwords**, and click **Create app password**.

c. Select permissions for your password: select **Read** permission for Repositories, and if you need to, add other permissions according to your environment. When you are done, click **Create**.

d. Copy the password you created, and paste it into the **Password** field in the JFrog platform.

5. Click **Save & Finish**.

6. Create a new virtual repository and under **Repositories**, select the remote repository you have created. You can use the virtual repository to resolve artifacts from your Bitbucket Cloud private registry.

7.17.2.2.1.6 | Proxy Bitbucket Server with Go

#### Go Proxy Best Practice: Add Mirror Repository

We highly recommend that you create a remote repository pointing to the default go mirror ([gocenter.io](https://gocenter.io)) and enter **Artifactory** as the **Git provider**. This way, Artifactory will be able to resolve cached public modules without querying the external registry, reducing the chance of rate limiting. For more information, see [Set Up Go Proxy Mirror Repositories](#).

To create a remote repository that serves as a caching proxy for Bitbucket Server, follow these steps:

1. Create a new remote repository and set **Go** as the **Package Type**.
2. Set the **Repository Key** value, and enter your Bitbucket URL in the **URL** field: for example, <https://git.acme.org/>.
3. In the **Go Settings** tab, set the **Git Provider** as **Bitbucket Server**.
4. In the **Advanced** tab, enter the username Bitbucket account under **Username**, and your token under **Password**. To create a token:
  - a. Go to your Bitbucket account, click your avatar in the top right corner, and select **Manage account**.

b. Go to **HTTP Access tokens**, and click **Create token**.

c. Give a label and select permissions for your password: select **Read** permission for Repositories, and if you need to, add other permissions according to your environment. When you are done, click **Create**.

d. Copy the token you created, and paste it into the **Password** field in the JFrog platform.

5. Click **Save & Finish**.

6. Create a new virtual repository and under **Repositories**, select the remote repository you have created. You can use the virtual repository to resolve artifacts from your Bitbucket Server private registry.

7.17.2.2.2 | [Work with GOSUMDB](#)

Go Client 1.13 introduced support for checksum verifications of modules against a central server. Artifactory is able to act as the central checksum server for the Go clients using it as the Go proxy. This will work without any additional configurations for remote sites pointing to GitHub or another Artifactory server.

It is possible to override the default central server used by Artifactory to provide the checksums by adding the property `artifactory.go.sumdb.url.override= https://...` to your Artifactory's system properties.

This feature can be disabled by adding the property `artifactory.go.sumdb.enabled=false` to your Artifactory's system properties.

Private GitHub repositories and some remote sites do not provide checksums. The Go client in those cases will complain about missing the checksum. You can control this feature on the client side with the `GOPRIVATE` and `GONOSUMDB` environment variables. See the [Go 1.13 Modules](#) environment variables for full details.

7.17.2.3 | [Set Up Virtual Go Repositories](#)

**Default Deployment Repository**

If you are publishing your Go builds to a Virtual Repository, make sure to set one of the local Go repositories that it aggregates as the Default Deployment Repository.

# JFrog Artifactory Documentation

## Displayed in the header

A Virtual Repository defined in Artifactory aggregates packages from both local and remote repositories.

This allows you to access both locally hosted Go packages and packages from remote proxied Go registries from a single URL defined for the virtual repository.

To create a virtual repository as a Go registry, in the **Administration** module, under **Repositories | Repositories | Virtual**, click **New Virtual Repository** and set **Go** to be its **Package Type**, and select the underlying local and remote repositories to include under the **Repositories** section.

### 7.17.2.3.1 | Advanced Configuration for Go Repositories

Some **Remote Import Paths** may use `go-import` meta tags on the remote repository response body to declare the location of a remote VCS root to follow. By default, this behavior is enabled, and Artifactory will follow these tags to download remote modules. To disable this uncheck the **Follow 'go-import' Meta Tags** checkbox.

Field	Description
Follow 'go-import' Meta Tags	When checked (default), Artifactory will automatically follow remote VCS roots in <code>go-import</code> meta tags to download remote modules.
'go-import' Allow List	An Allow List of Ant-style path patterns that determine which remote VCS roots Artifactory will follow to download remote modules from when presented with <code>go-import</code> meta tags in the remote repository response. By default, this is set to <code>**</code> which means that remote modules may be downloaded from any external VCS source.  For example, if you wish to limit remote <code>go-import</code> modules to only be downloaded from <code>github.com</code> , you should remove the default <code>**</code> pattern and replace it with <code>**/ github.com/**</code> .

### 7.17.3 | Use Go with Artifactory

Much of your work with Go and Artifactory is done through JFrog CLI, a thin client that wraps the Go client. To learn more, please refer to the [JFrog CLI User Guide](#).

#### Did you know?

JFrog CLI is, itself, written in Go.

### 7.17.3.1 | Resolve Go Projects

#### Resolving Go Only Via Local or Virtual Repositories

Artifactory only supports resolution of Go packages from **virtual Go repositories**. To resolve Go from other local or remote Go repositories, you need to aggregate them in a virtual Go repository.

`go.mod` is a metadata file that describes a Go package. It contains the package's module name and a list of its dependencies.

To allow successful resolution of a package, the Go client requires a corresponding `go.mod` file to be found in the same folder.

For example, in this Hello Worlds Go project, the `go.mod` file specifies a module `github.com/you/hello`, with a single dependency, `rsc.io/quote v1.5.2`.

#### go.mod

```
module github.com/you/hello
require rsc.io/quote v1.5.2
```

### 7.17.3.1.1 | Resolve Transitive Go Dependencies Locally

To fully resolve a Go project, each transitive dependency needs to have its corresponding `go.mod` file in place in the same folder where it resides. However, in many cases, the `go.mod` file does not exist. To resolve a Go project in this case, follow these steps:

1. Manually download all required dependencies from the internet to your local machine

## JFrog Artifactory Documentation Displayed in the header

2. Run your build while resolving dependencies locally

3. Publish your built package to Artifactory as described below.

From this point on, you can rebuild your project while resolving dependencies from Artifactory

### 7.17.3.2 | Build Go Packages

To build your Go projects, use JFrog CLI. JFrog CLI downloads the dependencies needed for the project from the internet and creates the corresponding `go.mod` file. When you later use JFrog CLI to publish your package to Artifactory, the `go.mod` file is uploaded alongside the package to Artifactory.

For details, please refer to the [JFrog CLI documentation](#).

### 7.17.3.3 | Publish Go Projects

The Go client works through a Git repository which hosts the Go package source code, and does not provide a way to directly publish packages to Artifactory.

To publish your package to Artifactory, use JFrog CLI as described in the [JFrog CLI documentation](#).

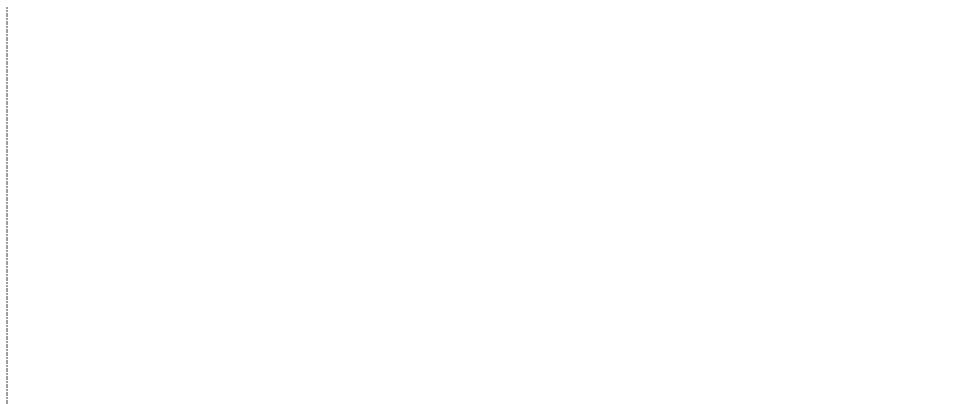
#### Default Deployment Repository

If you are publishing your Go builds to a Virtual Repository, make sure to set one of the local Go repositories that it aggregates as the [Default Deployment Repository](#).

### 7.17.4 | Limitations of Go in Artifactory

- Go remote repositories can only be accessed directly through a virtual repository. To resolve an artifact from a remote repository, you must [create a virtual repository](#), connect it to the remote repository, and use it to resolve the artifact.

## 7.18 | Hex Repositories

- 
- To start working with Hex, see [Get Started with Hex Repositories](#)
  - To configure your Hex package manager client to work through Artifactory, see [Set Me Up - Hex Client](#)

Hex is a package manager for the BEAM ecosystem. It supports any programming language that compiles to run on the BEAM VM, such as Elixir and Erlang. For more information, see the [Hex documentation](#).

Hex repository in Artifactory is a repository type specifically designed to handle Hex packages, which contain packages, resources, and metadata for use in Elixir and Erlang projects.

#### Benefits of Hex in Artifactory

- **Centralized Management:** Store and manage all your Hex packages in one place, making it easier to track dependencies and versions.
- **Consistent Builds:** Ensure developers and CI/CD systems use the same package versions, promoting reproducibility and consistency.
- **Improved Performance:** Cache packages from remote repositories to speed up builds and reduce external network calls.
- **Fine-Grained Access Control:** Control who can upload, download, and view packages, enhancing security and governance over your dependencies.
- **Reliability:** Local caching of remote packages ensures your build process continues even if external services, like Hex.pm, are unavailable.

#### Hex Supported Clients

Hex repository supports the following and has been tested. It lists the specific compatible versions to ensure smooth integration and optimal performance when working with Hex Repositories in Artifactory.

- **Hex:** Versions 2.1.1 and above
- **Elixir:** Versions 1.16 and above
- **Open Telecom Platform (OTP):** Versions 26.2.5 and above

### Limitations of Hex in Artifactory

The following are the limitations of Hex in Artifactory:

- **Mix Client:**
    - **Single Repository Limitation:** The Mix client allows only one repository to point to repo.hex.pm. As a result, you can only set one Artifactory endpoint as the default repository that connects to the public hex.pm registry. In the future, this will be resolved by the Hex Virtual Repository which will aggregate all packages
    - **Repository Naming:** The name of your self-hosted Hex repository in the Mix client must exactly match the full URL of the Artifactory remote repository.
  - **Hex Server:**
    - **Private Package Support:** Self-hosted Hex servers do not support private organizations or private packages. This feature is only available through the public hex.pm registry, which offers private hosted repositories.
  - **Hex Local Repository:**
    - We currently support using local repositories for storing first-party (internal) and second-party (partner) packages.
    - parsing any info related to docs, tests, etc.; from mix.exs is not supported.
    - mix hex.publish command to publish packages to local repository is not supported.
- Workaround:** Deploy Hex Packages to Hex Local Repository
- deploying packages under any other directory except **tarballs** is not supported. To learn more, refer to [Hex Repository Layout and Permission](#).
- **Hex Remote Repository:** We currently support using remote repositories for caching third-party packages from registries, such as hex.pm.
  - **Repository Browsing:** Artifactory does not support browsing Hex remote registries. The artifact tree only shows packages explicitly pulled into Artifactory, not the contents of the remote registry.

### 7.18.1 | Get Started with Hex Repositories

This topic outlines getting started with Hex repositories in Artifactory.

The following are the related topics:

- [Generate and Upload RSA Key Pair](#)
- [Create Hex Repository](#)
- [Set Me Up - Hex Client](#)

### 7.18.2 | Create Hex Repository

This section describes hex repositories in Artifactory and outlines how to create both local and remote Hex repositories in Artifactory. You can create Hex repositories to work locally using local repositories and proxy external Hex package sources using remote repositories.

#### Hex Repositories in Artifactory

In JFrog Artifactory, Hex repositories are used to store and manage hex packages written in the Elixir and Erlang programming languages. Artifactory supports two main types of Hex repositories, Local and Remote. These repositories serve different purposes in managing your Hex packages, whether they are internal to your organization or external dependencies.

##### Hex Local Repositories

A Hex Local Repository is a repository hosted within Artifactory that stores your Hex package and indexes it directly in Artifactory for storing first-party (internal) and second-party (partner) packages.

This type of repository is typically used for the following scenarios:

- **Storing Internal Packages:** When you create your hex packages (.tar) that you want to reuse or share across different projects within your organization.
- **Managing Version Control**
  - Storing different versions of your Hex packages and ensuring that they are accessible to other developers or CI/CD pipelines.
  - In a hex local repository, you can upload hex packages either manually or automatically through CI/CD tools, allowing your teams to consistently use the same package versions. Local repositories provide full control over the package storage, including access permissions, versioning, and caching.

# JFrog Artifactory Documentation

## Displayed in the header

### Hex Remote Repositories

A Hex Remote Repository in Artifactory acts as a proxy to remote sources of Hex packages, such as the public `Hex.pm` repository. By configuring a remote repository, Artifactory allows you to cache and fetch external and third-party Hex packages from these remote sources.

- **Proxying Remote Hex Registries:** Remote repositories in Artifactory act as proxies for remote Hex registries, such as the default `https://repo.hex.pm`, or other public or private repositories.
- **Package Caching and Access:** When a package is requested, Artifactory first checks its local cache. If the package is not found, it retrieves it from the remote registry via the internet and caches it locally for future use. This ensures faster access to previously fetched packages, even without an internet connection. Only the requested packages are cached, not the entire repository.
- **Cache Management and Limitations**
  - While resources in the cache can be removed, you cannot manually upload or push packages to a remote repository.
  - Remote repositories are ideal for integrating public dependencies into your development pipeline while maintaining control over which versions and packages are used.
  - Hex repositories in Artifactory allow streamlined package management, improve build reliability and performance, and ensure consistency across development environments. Whether managing internal packages or integrating external dependencies. You can maintain control, security, and efficiency throughout your Elixir development lifecycle.

The following are the related topics:

- [Create Hex Local Repository](#)
- [Create Hex Remote Repository](#)

#### 7.18.2.1 | Hex Repository Prerequisites

The Hex package manager is unique as it uses protobuf encoding for efficient binary sizing and performance. This means that every Hex package is encrypted by the server and decrypted by the client/proxy server using an RSA public key.

**Generate and upload RSA key pair to Artifactory:** These RSA keys will be assigned to your Hex repositories to encode the content for transfer to the Mix client.

For Artifactory to support the following you must meet the above prerequisite before proceeding with creating repositories:

- **Provide the Hex upstream registry public key**

This is entered during the remote repository creation and used to decrypt the package content being downloaded from the upstream registry to Artifactory.

#### Note

When using the `Hex.pm` public registry we auto-populate this value for you. For self-hosted Hex servers, you are required to provide the public key manually in the repository setup.

- **Place the Artifactory public key within your Hex Project**

This is required according to the Mix client specification to allow the Mix client to validate and decrypt the package content from Artifactory.

#### Note

We have streamlined this process by providing a quick download code snippet as part of our Set Me Up instructions.

#### 7.18.2.1.1 | Generate and Upload RSA Key Pair

This topic describes generating and uploading an RSA key pair to Artifactory. RSA keys are used for secure authentication when interacting with repositories, ensuring encrypted communication between clients and the artifactory.

Generate an RSA key pair, upload the public key to Artifactory, and configure it to streamline your development and deployment workflows.

To use the mix client with Artifactory, you must authenticate the client using an RSA key.

#### Understanding RSA Key Pairs in Hex Package Management



Hex uses Protobuf encoding for efficient binary sizing and performance. To download Hex packages, you must provide a public key in your project. This key allows the Mix client to validate the content from the registry and is used to encrypt and decrypt the data.

When using Artifactory as a middleman for Hex packages, you need two sets of RSA key pairs:

- **External Keys:** These keys are required to access the remote Hex registry (public, private, or self-hosted hex server). Each Hex registry has its own public and private key pair. You must enter the public key while creating the remote repository in Artifactory.
- **Internal Keys:** These keys allow the mix client to access Artifactory, which acts as a Hex server. You can set up the internal keys in Artifactory under **Security | Keys Management | RSA Key Pair**. Assign this key pair to the remote repository.

By using two sets of keys, we ensure secure communication between your project, Artifactory, and the external Hex registry.

# JFrog Artifactory Documentation

## Displayed in the header

To learn how to generate and upload an RSA key pair, refer to the [Setting Up RSA Keys Pairs](#)

To learn more, refer to the [RSA Key Pairs](#).

### Note

Once completed, proceed to [Create Hex Repository](#).

#### 7.18.2.2 | Create Hex Local Repository

This topic describes how to create a Hex local repository in Artifactory to securely deploy and manage your internal Hex packages. It provides instructions to create a local repository for efficient package management.

### Prerequisites

The Mix client requires that you generate and upload an RSA key pair to Artifactory. For more information, see [Generate and Upload RSA Key Pair](#).

**To create a local repository to upload hex packages, follow these steps:**

1. Click **Administration** tab, and then click **Repositories**.

2. Click **Local** from **Create a Repository** drop-down list.

3. Click **Hex** in the **Select Package Type** dialog.

4. Set **Repository Key** in the **New Local Repository** window.

5. Under **RSA Key Pair**, select the RSA key you would like to use from the drop-down menu.

This key is used for the protobuf encoding to communicate with your mix client.

6. (Optional) By default, Hex remote repositories are configured to point to the public Hex registry, and we automatically populate the upstream public key to decrypt Protobuf responses. If you wish to direct this remote repository to a self-hosted instance or a private Hex registry, update the URL and ensure you enter the corresponding public key of the registry's URL.

7. (Optional) For other **Basic** settings, refer to the [Basic Settings for Local Repositories](#).

8. (Optional) For **Advanced** settings, refer to the [Advanced Settings for Local Repositories](#).

9. Click **Create Local Repository**.

10. (Optional) If you are using a private Hex .pm registry, add your secret for your organization in the **Password/ Access Token** field.

7.18.2.3 | [Create Hex Remote Repository](#)

This topic describes how to create a Hex remote repository in Artifactory to resolve, cache, and manage external Hex packages. It provides instructions to create a remote repository that proxies remote sources, ensuring efficient access to and caching of both public and private Hex packages.

**Prerequisites**

The Mix client requires that you host your Hex repository RSA public key in your project folder to verify the index signature. For more information, see [Generate and Upload RSA Key Pair](#).

To create a remote repository to proxy a remote hex registry, follow these steps:

1. Click **Administration** tab, and then click **Repositories**.

2. Click **Remote** from **Create a Repository** drop-down list.

3. Click **Hex** in the **Select Package Type** dialog.

4. Set **Repository Key** in the **New Local Repository** window.

5. Under **RSA Key Pair**, select the RSA key you would like to use from the drop-down menu.

This key is used for the protobuf encoding to communicate with your mix client.

6. (Optional) By default, Hex remote repositories are configured to point to the public Hex registry, and we automatically populate the upstream public key to decrypt Protobuf responses. If you wish to direct this remote repository to a self-hosted instance or a private Hex registry, update the URL and ensure you enter the corresponding public key of the registry's URL.

7. (Optional) For other **Basic** settings, refer to the [Basic Settings for Remote Repositories](#).

# JFrog Artifactory Documentation

## Displayed in the header

8. (Optional) For **Advanced** settings, refer to the Advanced Settings for Remote Repositories.
9. Click **Create Remote Repository**.
10. (Optional) If you are using a private Hex .pm registry, add your secret for your organization in the **Password/ Access Token** field.

### 7.18.3 | Set Me Up - Hex Client

This section outlines how to set up project and mix client to work with Hex repositories in Artifactory. You can configure the project to work with Hex repositories to deploy and resolve packages using the local repository and resolve packages using the remote repositories.

#### Prerequisite

##### Download Public Key to Hex Project Folder

The following are the related topics:

- [Set Up Hex Local Repository](#)
- [Set Up Hex Remote Repository - hex.pm Hosted](#)
- [Set Up Hex Remote Repository - Self-hosted](#)

#### 7.18.3.1 | Download Public Key to Hex Project Folder

This topic describes how to download the public key to your project folder to work with Hex repositories in Artifactory. It provides instructions to download the public key using curl or wget.

The Mix client requires that you host your Hex repository RSA public key in your project folder to verify the index signature.

First, generate a user token using the set me up dialog. This token will auto-populate in all relevant placeholders within the set-me-up instructions.

##### To configure the Mix client, follow these steps:

Copy the publickey.pem key hosted in Artifactory to your Mix project folder using the following curl or Wget commands:

Refer to [View Set Me Up Instructions - Hex Repository](#) topic to view **Configure** code snippets.

- [Curl](#)

##### Note

Make sure to replace the placeholders in bold with your token, JFrog domain name, port, and JFrog repository key.

```
curl -u<USER>:<TOKEN> -o publickey.pem https://<YOUR_JFROG_DOMAIN>:<PORT>/artifactory/api/security/keypair/public/repositories/<REPOSITORY_NAME>
```

##### For example:

```
curl -uadmin:cmVmdGtu0jAx0j45NTgLX1YaFVCYXZY00NaY2dRUEJioEVpNFpT -o publickey.pem https://john.jfrog.io/artifactory/api/security/keypair/public/repositories/hex-remote
```

- [Wget](#)

##### Note

Make sure to replace the placeholders in bold with your token, JFrog domain name, port, and JFrog repository key.

```
wget -O publickey.pem https://<USER>:<TOKEN>@<YOUR_JFROG_DOMAIN>:<PORT>/artifactory/api/security/keypair/public/repositories/<REPOSITORY_NAME>
```

##### For example:

```
wget -O publickey.pem https://admin:cmVmdGtu0jAx0j45NTgLX1YaFVCYXZY00NaY2dRUEJioEVpNFpT@john.jfrog.io/artifactory/api/security/keypair/public/repositories/hex-remote
```

### 7.18.3.2 | Set Up Hex Local Repository

This section outlines how to set up hex local repository in Artifactory. You can configure the project and work with Hex local repositories to deploy and resolve packages.

#### Prerequisite

##### Download Public Key to Hex Project Folder

# JFrog Artifactory Documentation Displayed in the header

The following are the related topics:

- Configure Mix Client with Hex Local Repository
- Deploy Hex Packages to Hex Local Repository
- Resolve Hex Packages from Hex Local Repository

## 7.18.3.2.1 | Configure Mix Client with Hex Local Repository

This topic describes how to configure Hex local repository to deploy and resolve packages using Hex local repositories in Artifactory. It provides instructions to configure the mix client to deploy and resolve packages via Hex local repositories.

Connect your Mix client to the hex local repository.

### Prerequisite

Download Public Key to Hex Project Folder

To set up your Mix client pointing to Hex local repository, use the following command from your project folder:

Refer to View Set Me Up Instructions - Hex Repository topic to view **Configure** code snippets.

### Note

Make sure to replace the placeholders in bold with your token, JFrog domain name, port, and JFrog repository key. These placeholders are auto-populated if you copy the Set Up A Hex Client Configure code snippets from the UI.

Run the following command in the Hex project directory to add your Artifactory Local repository in the mix client:

```
mix hex.repo add <REPOSITORY_NAME> https:<YOUR_JFROG_DOMAIN>:<PORT>/artifactory/api/hex/<REPOSITORY_NAME> --auth-key "Bearer <TOKEN>" --public-key ./publickey.pem
```

### For example:

```
mix hex.repo add hc-hex-local https://hexdemotest.jfrogdev.org/artifactory/api/hex/hc-hex-local --auth-key "Bearer cmVmdGtu0jAx0jE3Njc3MDUwOTg6dFFCMj1ZTj12SkGTUZadUQyelBv" --public-key ./publickey.pem
```

### Note

Make sure that the `publickey.pem` key is pointing to the correct location in your project.

## 7.18.3.2.2 | Deploy Hex Packages to Hex Local Repository

This topic describes how to deploy hex packages to Hex local repositories in Artifactory. It provides instructions to deploy hex packages via API to resolve them in your organization's development activities.

### Prerequisite

Download Public Key to Hex Project Folder

To deploy hex packages via API, follow these steps:

1. Refer to View Set Me Up Instructions - Hex Repository topic to view **Configure and Deploy** code snippets.
2. Copy the code snippet and paste it into the CLI.

For example, to deploy a hex package into a repository called `hex-local`, you could use the following:

```
curl -u<USERNAME>:<PASSWORD> -XPUT http://localhost:8080/artifactory/hex-local/tarballs -T <HEX_PACKAGE_TAR_FILE_PATH>
```

where `<HEX_PACKAGE_TAR_FILE_PATH>` specifies the path from the repository root to the deploy folder.

## 7.18.3.2.3 | Resolve Hex Packages from Hex Local Repository

This topic describes how to resolve the packages deployed in Hex local repositories in Artifactory. It provides instructions to resolve the packages for the organization's development activities.

### Prerequisite

Download Public Key to Hex Project Folder

To resolve a Hex local package, follow these steps:

1. Refer to View Set Me Up Instructions - Hex Repository topic to view **Resolve** code snippets.
2. Add the package names, versions and repository names to your `mix.exs` file:

### Note

Make sure to replace the placeholders in bold with the package name, version, and repository name.

```
defp deps do
  [
    {:<PACKAGE_NAME>, "<PACKAGE_VERSION>", repo: "<REPOSITORY_NAME>"}
  ]
```

### For example:

```
defp deps do
  [
```

# JFrog Artifactory Documentation Displayed in the header

```
{:jason, "1.4.4", repo: "hex-local"}  
]
```

3. Run the following command:

```
mix deps.get
```

## 7.18.3.3 | Set Up Hex Remote Repository - hex.pm Hosted

This section outlines how to set up hex remote repository in Artifactory. You can configure the project and work with Hex remote repositories to resolve public and private packages.

### Prerequisite

[Download Public Key to Hex Project Folder](#)

The following are the related topics:

- [Configure Hex.pm - Hosted](#)
- [Resolve Hex Public Package](#)
- [Resolve Hex Private Package](#)

### 7.18.3.3.1 | Configure Hex.pm - Hosted

This topic describes how to configure hex.pm hosted to resolve packages using Hex remote repositories in Artifactory. It provides instructions to configure the mix client to resolve packages via Hex remote repositories from the public-hosted servers.

Connect your Mix client to the public registry or a private organization within hex.pm

### Prerequisite

[Download Public Key to Hex Project Folder](#)

To set up your Mix client using a Hex remote repository pointing to hex.pm, use the following command from your project folder:

Refer to [View Set Me Up Instructions - Hex Repository](#) topic to view **Configure** code snippets.

### Note

Make sure to replace the placeholders in bold with your token, JFrog domain name, port, and JFrog repository key. These placeholders are auto-populated if you copy the Set Up A Hex Client Configure code snippets from the UI.

```
mix hex.repo set hexpm --url http://<YOUR_JFROG_DOMAIN>:<PORT>/artifactory/api/hex/<REPOSITORY_NAME> --auth-key "Bearer <TOKEN>" --public-key ./publickey.pem
```

### For example:

```
mix hex.repo set hexpm --url https://john.jfrog.io/artifactory/api/hex/hex-remote --auth-key "Bearer cmVmdGtu0jAx0j45NTg2Nzg3NDA6RnkKLX1YafVCYXYZOONaY2dRUEJioEVpNFpT" --public-key ./publickey.pem
```

### Note

Make sure that the `publickey.pem` key is pointing to the correct location in your project.

### 7.18.3.3.2 | Resolve Hex Package from hex.pm

This section outlines how to resolve packages using Hex remote repositories in Artifactory. You can resolve packages via Hex remote repositories from the hosted and self-hosted servers.

To resolve a Hex package from Artifactory depends if you are pulling from the Hex public registry or from a specific private repository in hex.pm.

The following are the related topics:

- [Resolve Hex Public Package](#)
- [Resolve Hex Private Package](#)

### 7.18.3.3.2.1 | Resolve Hex Public Package

This topic describes how to resolve packages using Hex remote repositories in Artifactory. It provides instructions to resolve packages using Hex remote repository that points to the official Hex.pm registry, enabling access to public Hex packages.

You can resolve public hex.pm packages to your mix project. For more information about Mix projects, see the [Mix documentation](#).

### Prerequisite

[Download Public Key to Hex Project Folder](#)

**To resolve a Hex public package, follow these steps:**

1. Refer to [View Set Me Up Instructions - Hex Repository](#) topic to view **Resolve** code snippets.
2. Add the package names and versions to your `mix.exs` file:

### Note

Make sure to replace the placeholders in bold with the package name and version.

```
defp deps do  
[
```

# JFrog Artifactory Documentation

## Displayed in the header

```
{:<PACKAGE_NAME>, "<PACKAGE_VERSION>"}  
]
```

### For example:

```
defp deps do  
[  
  {:jason, "1.4.4"}  
]
```

3. Run the following command:

```
mix deps.get
```

#### 7.18.3.3.2.2 | Resolve Hex Private Package

This topic describes how to resolve packages using Hex remote repositories in Artifactory. It provides instructions to resolve packages using Hex remote repository that points to the official Hex.pm registry, enabling access to private Hex packages in your organization.

To resolve a Hex private package, follow these steps:

#### Prerequisite

##### Download Public Key to Hex Project Folder

1. Refer to [View Set Me Up Instructions - Hex Repository](#) topic to view **Resolve** code snippets.
2. Add the package names, versions and organization names to your mix.exs file:

#### Note

Make sure to replace the placeholders in bold with the package name, version, and Hex organization name.

```
defp deps do  
[  
  {:<PACKAGE_NAME>, "<PACKAGE_VERSION>", organization: "<HEX_ORGANIZATION_NAME>"}  
]
```

### For example:

```
defp deps do  
[  
  {:jason, "1.4.4", organization: "acme"}  
]
```

3. Run the following command:

```
mix deps.get
```

#### 7.18.3.4 | Set Up Hex Remote Repository - Self-hosted

This section outlines how to set up hex remote repository in Artifactory. You can configure the project and work with Hex remote repositories to resolve packages.

#### Prerequisite

##### Download Public Key to Hex Project Folder

The following are the related topics:

- [Configure Hex Server \(Self-hosted\)](#)
- [Resolve Hex Self-hosted Package](#)

#### 7.18.3.4.1 | Configure Hex Server (Self-hosted)

This topic describes how to configure hex.pm self-hosted to resolve packages using Hex remote repositories in Artifactory. It provides instructions to configure the mix client to resolve packages via Hex remote repositories from the self-hosted servers.

Connect your Mix client to a private organization within hex.pm

#### Prerequisite

##### Download Public Key to Hex Project Folder

**To configure your Mix client for a private self-hosted Hex registry that points to hex.pm, run the following command from your project folder:**

Refer to [View Set Me Up Instructions - Hex Repository](#) topic to view **Configure** code snippets.

#### Note

Make sure to replace the placeholders in bold with your token, JFrog domain name, port, and JFrog repository key. These placeholders are auto-populated if you copy the Set Up A Hex Client Configure code snippets from the UI.

```
mix hex.repo add <REPOSITORY_KEY> http://<HOST>:<PORT>/artifactory/api/hex/<REPOSITORY_KEY> --auth-key "Bearer <TOKEN>" --public-key ./publickey.pem
```

### For example:

```
mix hex.repo add hex-test https://john.jfrog.io/artifactory/api/hex/hex-remote --auth-key "Bearer cmVmGtu0jAx0j45NTgzNzg3NDA6RnkKLX1YaFVCYXZY00NaY2dRUEJioEVpNfpT" --public-key ./publickey.pem
```

### Note

Make sure that the publickey.pem key is pointing to the correct location in your project.

#### 7.18.3.4.2 | Resolve Hex Self-hosted Package

This topic describes how to resolve packages using Hex remote repositories in Artifactory. It provides instructions to resolve packages using Hex remote repository that points to the self-hosted server.

This section describes how to resolve a hex self-hosted package.

### Prerequisite

Download Public Key to Hex Project Folder

To resolve a Hex self-hosted package, follow these steps:

1. Refer to View Set Me Up Instructions - Hex Repository topic to view **Resolve** code snippets.
2. Add the package names, versions and repository names to your mix.exs file:

### Note

Make sure to replace the placeholders in bold with the package name, version, and repository name.

```
defp deps do
  [
    {:<PACKAGE_NAME>, "<PACKAGE_VERSION>", repo: "<REPOSITORY_NAME>"}
  ]
```

### For example:

```
defp deps do
  [
    {:jason, "1.4.4", repo: "hex-remote"}
  ]
```

3. Run the following command:

```
mix deps.get
```

#### 7.18.4 | Advanced Topics - Hex Repository

This section outlines all the other topics related to Hex repositories in Artifactory. You can learn JPD authentication, repository layout and permission, view set-me-up instructions, deploy hex packages via UI, and manage hex repositories and packages.

The following are the related topics:

- Hex Repository Layout and Permission
- Manage Hex Repository
- Manage Hex Packages
- Deploy Hex Packages via UI
- View Set Me Up Instructions - Hex Repository
- Hex Supported Commands

#### 7.18.4.1 | Hex Repository Layout and Permission

This topic describes the Hex local and remote repositories layout and permissions you can apply.

- Hex Local Repository Layout
- Hex Remote Repository Layout

### Hex Local Repository Layout

The following is the Hex Local Repository Layout.

### Note

All the uploaded tar files must be under the tarballs directory as tarballs/[name]-[version].tar.

```
<repository name>
  |--- names
  |--- versions
  |--- installs
  |   |--- hex-1.x.csv
  |--- packages
  |   |--- <package-name>
  |--- tarballs
  |   |--- <package-name>-<package-version>.tar
```

- **repository name** - Name of the repository
- **names** - Name details of a package
- **versions** - Version details of a package
- **installs** - Metadata uploaded for the package
- **packages** - Metadata of packages uploaded in the local
- **tarballs** - Packages uploaded in the local repository

### Hex Remote Repository Layout

```
<REPOSITORY_NAME>
├── installs
│   |-- hex-1.x.csv
├── packages
│   |-- <package-name>
└── tarballs
    |-- <package-name>-<package-version>.tar
```

- **Installs** - Metadata pulled from the remote
- **packages** - The package metadata pulled from the remote
- **tarballs** - Redistributable binaries for each package pulled from the remote

### Permissions on Hex Remote Repository Directories

You can provide granular permissions for users to specific packages. To limit access, apply permissions to each of the layout folders.

For example, to exclude packages that start with abc in the hex repository, add Exclude Patterns for each layout folder as:

- installs/abc\*
- tarballs/abc\*
- packages/abc\*

To learn more, refer to the following:

- [Include/Exclude Patterns](#)
- [Add Repositories](#)

#### 7.18.4.2 | Manage Hex Repository

This section outlines how to edit repository configurations, delete local and remote Hex repositories, and recalculate the index of a Hex local repository in Artifactory. You can edit existing repositories to properly configure them to meet your needs, delete unintended repositories, and ensure the local repository index is current for efficient package management.

The following are the related topics:

- [Recalculate Index of Hex Local Repository](#)
- [Edit Hex Remote Repository](#)
- [Delete Hex Remote Repository](#)

#### 7.18.4.2.1 | Recalculate Index of Hex Local Repository

The topic describes how to recalculate the index of a hex local repository. In Artifactory, once the repository is set, the system will index artifacts and calculate the corresponding metadata for every package uploaded, which optimizes performance when resolving artifacts.

To recalculate a hex local repository index, follow these steps:

1. Click **Administration** tab, and then click **Repositories**.
2. Click **Local** in the **Repositories** window.
3. Hover the mouse pointer over the repository you want to delete, click **more** icon, and then click **Recalculate Index**.



To learn more, refer to [Repositories Index in Artifactory](#).

#### 7.18.4.2.2 | Edit Hex Repository

# JFrog Artifactory Documentation

## Displayed in the header

This section outlines how to edit Hex repositories in Artifactory. You can edit repository settings for both local and remote Hex repositories, ensuring they are properly configured to meet your needs.

The following are the related topics:

- [Edit Hex Local Repository](#)
- [Edit Hex Remote Repository](#)

7.18.4.2.2.1 | [Edit Hex Local Repository](#)

This topic describes how to edit Hex local repositories in Artifactory. It provides instructions to edit repository settings for Hex local repositories, ensuring they are properly configured to meet your needs.

**To edit a hex local repository, follow these steps:**

1. Click **Administration** tab, and then click **Repositories**.
2. Click **Local** in the **Repositories** window.
3. Click the hex local repository you want to edit from the list.



4. Edit the fields you want to make changes.



5. Click **Save**.

7.18.4.2.2.2 | [Edit Hex Remote Repository](#)

This topic describes how to edit Hex remote repositories in Artifactory. It provides instructions to edit repository settings for Hex remote repositories, ensuring they are properly configured to meet your needs.

**To edit a hex remote repository, follow these steps:**

1. Click **Administration** tab, and then click **Repositories**.
2. Click **Remote** in the **Repositories** window.
3. Click the hex remote repository you want to edit from the list.



4. Edit the fields you want to make changes.

5. Click **Save**.

7.18.4.2.3 | [Delete Hex Repository](#)

This section outlines how to delete both local and remote Hex repositories in Artifactory. You can delete unintended Hex repositories.

The following are the related topics:

- [Delete Hex Local Repository](#)
- [Delete Hex Remote Repository](#)

7.18.4.2.3.1 | [Delete Hex Local Repository](#)

This topic describes how to delete Hex local repositories in Artifactory. You can delete unintended Hex local repositories.

**To delete a hex local repository, follow these steps:**

1. Click **Administration** tab, and then click **Repositories**.
2. Click **Local** in the **Repositories** window.
3. Hover the mouse pointer over the repository you want to delete, click **more** icon, and then click **Delete**.

7.18.4.2.3.2 | [Delete Hex Remote Repository](#)

This topic describes how to delete Hex remote repositories in Artifactory. You can delete unintended Hex remote repositories.

**To delete a hex remote repository, follow these steps:**

1. Click **Administration** tab, and then click **Repositories**.
2. Click **Remote** in the **Repositories** window.
3. Hover the mouse pointer over the repository you want to delete, and then click **Delete** icon.

7.18.4.3 | [Manage Hex Packages](#)

# JFrog Artifactory Documentation

## Displayed in the header

This section outlines how to manage hex packages. You can search and list hex packages, view hex build info, view individual hex package information, and clean up hex local repositories in Artifactory.

The following are the related topics:

- [Search Hex Packages](#)
- [List Hex Package Versions/Tags](#)
- [View Hex BuildInfo](#)
- [View Individual Hex Package Information](#)
- [Clean Up Hex Local Repository Cache](#)

7.18.4.3.1 | [Search Hex Packages](#)

This topic describes how to search Hex packages in Artifactory. It provides instructions to search packages using a number of parameters.

You can search for Hex packages by name using the [Artifact Package Search](#) or through the REST API.

### Search Hex Packages via UI

Artifactory supports a variety of ways to search for artifacts. For details, please refer to [Browse and Search Artifacts](#). However, the packages may not be available immediately after being published, for the following reasons:



- When publishing packages to a local repository, Artifactory calculates the search index asynchronously and will wait to index the newly published packages.
- 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.

### Search Hex Packages via API

Search Artifactory using REST API. Refer to [Artifact Search API](#).

#### Tip

Artifactory annotates each deployed or cached package with the following properties:

`hex.name` and `hex.version`.

You can use Property Search to search for hex packages according to their name or version.

7.18.4.3.2 | [List Hex Package Versions/Tags](#)

This topic describes how to list Hex package versions in Artifactory. It provides references to work with listing package versions.

The list displays information about the package versions.

To learn more about list versions, refer to the [Viewing Package Information](#).

7.18.4.3.3 | [View Hex BuildInfo](#)

This topic describes how to view Hex Buildinfo in Artifactory. It provides references to view Buildinfo via UI and API.

Build-info is all the information collected by the build agent, which includes details about the build. The build-info includes a list of project modules, artifacts, dependencies, environment variables and more. When using one of the JFrog clients to build the code, the client can collect the build-info and publish it to Artifactory. When the build-info is published to Artifactory, all the published details become visible in the Artifactory UI.

### View Hex BuildInfo via UI

Refer to the [View Build Number Information](#) documentation.

### View Hex BuildInfo via API

Refer to the [Build Info REST API](#) documentation.

7.18.4.3.4 | [View Individual Hex Package Information](#)

This topic describes how to view individual hex package information in Artifactory. It provides instructions to view selected hex package metadata and their contents.

**To view hex package info, follow these steps:**

# JFrog Artifactory Documentation

## Displayed in the header

1. Click **Application** tab, and then click **Artifacts** under **Artifactory**.
2. Drill down in the **Tree Browser**, select the **tar.gz** file you want to inspect, and view the **Package Info** from the **Hex Info** tab.

### 7.18.4.3.5 | Clean Up Hex Local Repository Cache

The Hex client saves caches of downloaded packages and their JSON metadata responses (called `.cache.json`).

The JSON metadata cache files contain the Hex package metadata.

We recommend removing the Hex package caches 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://repo.hex.pm>.

### 7.18.4.4 | Deploy Hex Packages via UI

This topic describes how to deploy hex packages via UI to Hex local repositories in Artifactory to resolve them in your organization's development activities.

Once you have configured your local machine to install packages from your hex local repository, you can also deploy hex packages to the same repository using the REST API.

**To deploy hex packages via UI, follow these steps:**

1. Click **Application** tab, and then click **Artifacts** under **Artifactory**.
2. Locate the repository you want to configure in the Artifact tree browser, and then click **Deploy**.

Packages can be deployed in **Single** or **Multiple**.

3. Do one of the following based on your deployment type:

- Single Deploy

Drop the file or Select the file, and then click **Deploy**.

- Multiple Deploy

Drop the file or Select the file, and then click **Deploy**.

#### 7.18.4.5 | View Set Me Up Instructions - Hex Repository

This topic describes how to view Hex repository set-me-up instructions. It provides instructions, for the created Hex repositories, to view helpful code snippets from the UI to resolve Hex packages using the Mix client, the hex.pm build tool.

To view Hex Set Me UP instructions based on your repository selection, follow these steps:

1. Click **Application** tab, and then click **Artifacts** under **Artifactory**.

2. Locate the repository you want to configure in the Artifact tree browser, and then click **Set Me Up**.

The Set Me Up right pane appears.

**Note**

Alternatively, you can choose the repository from the **Repository** drop-down list under **Set UP A Hex Client** modal.

3. Enter your JFrog account password to generate a token, and then click **Generate Token & Create Instructions**.

4. Click one of the following tabs as applicable:

Tabs	Description								
Configure	Configure project and mix client to work with Hex repositories in Artifactory. You can configure the project to work with Hex repositories to deploy and resolve packages using the local repository and resolve packages using remote repositories.								
Deploy	This topic describes how to deploy hex packages to Hex local repositories in Artifactory. It provides instructions to deploy hex packages via API to resolve them in your organization's development activities.								
Resolve	<table border="1"><thead><tr><th>Type</th><th>Description</th></tr></thead><tbody><tr><td>Hex Local Repository</td><td>This topic describes how to resolve the packages deployed in Hex local repositories in Artifactory. It provides instructions to resolve the packages for the organization's development activities.</td></tr><tr><td>Resolve Hex Package from hex.pm</td><td><ul style="list-style-type: none"><li>• <b>Public packages</b> This topic describes how to resolve packages using Hex remote repositories in Artifactory. It provides instructions to resolve packages using Hex remote repository that points to the official Hex.pm registry, enabling access to public Hex packages.</li><li>• <b>Private packages (Organization)</b> This topic describes how to resolve packages using Hex remote repositories in Artifactory. It provides instructions to resolve packages using Hex remote repository that points to the official Hex.pm registry, enabling access to private Hex packages in your organization.</li></ul></td></tr><tr><td>Self-hosted hex server</td><td>This topic describes how to resolve packages using Hex remote repositories in Artifactory. It provides</td></tr></tbody></table>	Type	Description	Hex Local Repository	This topic describes how to resolve the packages deployed in Hex local repositories in Artifactory. It provides instructions to resolve the packages for the organization's development activities.	Resolve Hex Package from hex.pm	<ul style="list-style-type: none"><li>• <b>Public packages</b> This topic describes how to resolve packages using Hex remote repositories in Artifactory. It provides instructions to resolve packages using Hex remote repository that points to the official Hex.pm registry, enabling access to public Hex packages.</li><li>• <b>Private packages (Organization)</b> This topic describes how to resolve packages using Hex remote repositories in Artifactory. It provides instructions to resolve packages using Hex remote repository that points to the official Hex.pm registry, enabling access to private Hex packages in your organization.</li></ul>	Self-hosted hex server	This topic describes how to resolve packages using Hex remote repositories in Artifactory. It provides
Type	Description								
Hex Local Repository	This topic describes how to resolve the packages deployed in Hex local repositories in Artifactory. It provides instructions to resolve the packages for the organization's development activities.								
Resolve Hex Package from hex.pm	<ul style="list-style-type: none"><li>• <b>Public packages</b> This topic describes how to resolve packages using Hex remote repositories in Artifactory. It provides instructions to resolve packages using Hex remote repository that points to the official Hex.pm registry, enabling access to public Hex packages.</li><li>• <b>Private packages (Organization)</b> This topic describes how to resolve packages using Hex remote repositories in Artifactory. It provides instructions to resolve packages using Hex remote repository that points to the official Hex.pm registry, enabling access to private Hex packages in your organization.</li></ul>								
Self-hosted hex server	This topic describes how to resolve packages using Hex remote repositories in Artifactory. It provides								

Tabs	Description
Type	Description
	instructions to resolve packages using Hex remote repository that points to the self-hosted server.

#### 7.18.4.6 | Hex Supported Commands

This topic describes the hex supported commands in Artifactory.

- mix deps.get --verbose
- mix deps.update --all
- mix hex.outdated <packageName>
- mix hex.repo list

### 7.19 | Hugging Face Repositories

Artifactory's integration with Hugging Face allows you to create a single system of record for ML models that brings ML/AI development in line with your existing SSC, using Artifactory to integrate machine learning into your stack.

#### 7.19.1 | Main Features of Hugging Face in Artifactory

##### Cache Hugging Face models and datasets on remote repositories

Proxy and cache machine learning models, datasets, and related files from Hugging Face on Artifactory using remote repositories to provide fast, consistent, and reliable access to members across your organization.

##### Store models and datasets in local repositories

Protect your proprietary information by deploying models, datasets, and related files to Artifactory local repositories, giving you fine-grain control of the access to your models and datasets.

##### Run Xray license scans on models

Secure your machine learning model lifecycle by scanning the licenses of machine learning models and scanning for malicious models.

##### Note

Hugging Face repositories are supported from Artifactory version 7.77.1, and deploying and resolving datasets is supported from Artifactory version 7.90.x. Security scanning of datasets is coming soon.

Spaces are not currently supported.

#### 7.19.2 | Create Hugging Face Repositories

This section describes how to create Hugging Face Repositories. It includes the following guides:

- Set Up Local Hugging Face Repositories
- Set Up Remote Hugging Face Repositories

#### 7.19.2.1 | Hugging Face Repository Structure

Hugging Face repositories are structured using two types of versions:

- Integration version: these versions are similar to branches in a Git repository. In local repositories the only integration version is Main, and in remote repositories, the integration version is created according to the requested branch.
- Release versions: these versions can be tags, commits, or revisions.

The Hugging Face repository structure for local repositories is as follows:

# JFrog Artifactory Documentation

## Displayed in the header

```
└── hf-local
    ├── models
    │   ├── org (optional)
    │   │   ├── model-name
    │   │   │   ├── main
    │   │   │   ├── timestamp
    │   │   │   │   ├── .jfrog_huggingface_dataset_info.json
    │   │   │   │   ├── file1
    │   │   │   │   ├── file2
    │   │   │   ├── timestamp
    │   │   │   │   ├── .jfrog_huggingface_dataset_info.json
    │   │   │   │   ├── file1
    │   │   │   │   ├── file2
    │   │   │   ├── timestamp (server)
    │   │   │   │   ├── .jfrog_huggingface_dataset_info.json
    │   │   │   │   ├── file1
    │   │   │   │   ├── file2
    │   │   │   ├── v1
    │   │   │   ├── timestamp
    │   │   │   │   ├── .jfrog_huggingface_dataset_info.json
    │   │   │   │   ├── file1
    │   │   │   │   ├── file2
    │   ├── datasets
    │   │   ├── org (optional)
    │   │   │   ├── dataset-name
    │   │   │   │   ├── main
    │   │   │   │   ├── timestamp
    │   │   │   │   │   ├── .jfrog_huggingface_dataset_info.json
    │   │   │   │   │   ├── file1
    │   │   │   │   │   ├── file2
    │   │   │   │   ├── timestamp
    │   │   │   │   │   ├── .jfrog_huggingface_dataset_info.json
    │   │   │   │   │   ├── file1
    │   │   │   │   │   ├── file2
    │   │   │   │   ├── timestamp (server)
    │   │   │   │   │   ├── .jfrog_huggingface_dataset_info.json
    │   │   │   │   │   ├── file1
    │   │   │   │   │   ├── file2
    │   │   │   ├── v1
    │   │   │   ├── timestamp
    │   │   │   │   ├── .jfrog_huggingface_dataset_info.json
    │   │   │   │   ├── file1
    │   │   │   │   ├── file2
```

Note the following:

- When resolving (downloading) a model or dataset without specifying a revision, the latest model or dataset version will be saved with the timestamp under the 'main' branch in the model or dataset name folder. Previous versions are retrievable using the SHA1 value, which is stored as the property `huggingfacem1.generated.revision.sha1` in the `.jfrog_huggingface_model_info.json` file.
- When deploying (uploading) a model or dataset with a specific version, it will be saved with a timestamp under that version name in the model or dataset's name folder. If the same model or dataset has been deployed already with the same version number, the new deployment will overwrite it, and the previous deployment will not be retrievable using the Artifactory-generated SHA1 value.

### Note

For Artifactory to overwrite the model, go into **Administration > User Management > Permissions**, and verify that the **Delete/Overwrite** permission is enabled. Otherwise, when trying to deploy a model that has been deployed before with the same version number, Artifactory will return a 409 error.

Starting from Artifactory version 7.77, the repository structure for remote repositories is as follows:

```
└── hf-remote
    ├── models
    │   ├── org (optional)
    │   │   ├── model-name
    │   │   │   ├── main
    │   │   │   ├── .latest_huggingface_model_info.json
    │   │   │   ├── timestamp 1
    │   │   │   │   ├── .jfrog_huggingface_model_info.json
    │   │   │   │   ├── file1
    │   │   │   │   ├── file2
    │   │   │   ├── timestamp 2
    │   │   │   │   ├── .jfrog_huggingface_model_info.json
    │   │   │   │   ├── file1
    │   │   │   │   ├── file2
    │   │   │   ├── timestamp 3
    │   │   │   │   ├── .jfrog_huggingface_model_info.json
    │   │   │   │   ├── file1
    │   │   │   │   ├── file2
    │   │   │   ├── v1
    │   │   │   ├── .latest_huggingface_model_info.json
    │   │   │   ├── timestamp 1
    │   │   │   │   ├── .jfrog_huggingface_model_info.json
    │   │   │   │   ├── file1
    │   │   │   │   ├── file2
    │   │   │   ├── timestamp 2
    │   │   │   │   ├── .jfrog_huggingface_model_info.json
    │   │   │   │   ├── file1
    │   │   │   │   ├── file2
    │   │   │   ├── original hash
    │   │   │   ├── .latest_huggingface_model_info.json
    │   │   │   ├── timestamp 1
    │   │   │   │   ├── .jfrog_huggingface_model_info.json
    │   │   │   │   ├── file1
```

```
└── file2
    ├── datasets
    │   └── org (optional)
    │       └── dataset-name
    │           ├── main
    │           │   ├── .latest_huggingface_dataset_info.json
    │           │   ├── timestamp 1
    │           │       ├── .jfrog_huggingface_dataset_info.json
    │           │       ├── file1
    │           │       ├── file2
    │           ├── timestamp 2
    │           │   ├── .jfrog_huggingface_dataset_info.json
    │           │   ├── file1
    │           │   ├── file2
    │           ├── timestamp 3
    │           │   ├── .jfrog_huggingface_dataset_info.json
    │           │   ├── file1
    │           │   ├── file2
    │           └── v1
    │               ├── .latest_huggingface_dataset_info.json
    │               ├── timestamp 1
    │                   ├── .jfrog_huggingface_dataset_info.json
    │                   ├── file1
    │                   ├── file2
    │               ├── timestamp 2
    │                   ├── .jfrog_huggingface_dataset_info.json
    │                   ├── file1
    │                   ├── file2
    │               └── original hash
    │                   ├── .latest_huggingface_dataset_info.json
    │                   ├── timestamp 1
    │                       ├── .jfrog_huggingface_dataset_info.json
    │                       ├── file1
    │                       ├── file2
```

Note the following:

- When resolving (downloading) a model or dataset without specifying a revision, the latest model or dataset version will be saved with the timestamp under the 'main' branch in the model or dataset's name folder. Previous versions are retrievable using the SHA1 value, which is stored as the property `huggingfacem1.generated.revision.sha1` on the `.jfrog_huggingface_model_info.json` file.
- When resolving (Downloading) a model or dataset with a specific version using a commit hash or tag, it will be saved with a timestamp under the requested revision in the model or dataset's name folder.

### 7.19.2.2 | Hugging Face Naming Limitations

Please structure the names of your organizations, model names, and revision identifiers according to the Hugging Face limitations when deploying models to local Hugging Face repositories:

#### Organization name limitations:

- The name can only contain letters, numbers, and hyphens (-)
- The name cannot start or end with a hyphen, or use consecutive hyphens (--)
- The name cannot contain only numbers

#### Model name limitations:

- The name can only contain letters, numbers, hyphens (-), underscores (\_), and periods (.)

#### Revision identifier limitations:

- The name can only contain letters, numbers, hyphens (-), and periods (.)
- The name cannot contain spaces or control characters (^, \$, etc.)

### 7.19.2.3 | Set Up Local Hugging Face Repositories

A local Hugging Face repository is where you deploy and host your internal Hugging Face resources. It is, in effect, a Hugging Face registry able to host collections of Hugging Face models. 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 create a local Hugging Face repository:

- From the **Administration** module, select **Repositories** | **Repositories** | **Local**.
- Click **New Local Repository** and select **Hugging Face** from the **Select Package Type** dialog.
- Set the **Repository Key** value.

### 7.19.2.4 | Set Up Remote Hugging Face Repositories

Artifactory supports proxying remote Hugging Face registries through remote repositories. A Remote Repository in Artifactory serves as a caching proxy for a registry managed at Hugging Face Hub.

### Note

Hugging Face remote repositories only supports the following URL, populated by default: <https://huggingface.co>

1. From the Administration module, select **Repositories | Repositories | Remote**.
2. Click **New Remote Repository** and select **Hugging Face** from the **Select Package Type** dialog.
3. In the **Basic** tab, set the **Repository Key** value.

### 7.19.3 | Set Up Hugging Face SDK To Work With Artifactory

This section describes how to set up Hugging Face SDK to work with Artifactory. It includes the following guides:

- Configure Hugging Face SDK to Work With Artifactory
- Deploy Hugging Face Models and Datasets
- Resolve Hugging Face Models and Datasets

#### 7.19.3.1 | Configure Hugging Face SDK to Work With Artifactory

This topic describes how to configure Hugging Face SDK to work with Artifactory.

To configure the Hugging Face SDK to work with Artifactory:

1. Go to the JFrog Platform **Artifacts** page, select your Hugging Face repository, and click **Set Me Up**.
2. To create a Hugging Face token, enter your JFrog Platform password, and click **Generate Token & Create Instructions**.
3. Connect your HuggingFaceML repository to the Hugging Face SDK in Artifactory using the following command:

```
export HF_HUB_ETAG_TIMEOUT=86400
export HF_HUB_DOWNLOAD_TIMEOUT=86400
export HF_ENDPOINT=https://<YOUR_JFROG_DOMAIN>/artifactory/api/huggingfaceml/<REPOSITORY_NAME>
```

For example:

```
export HF_HUB_ETAG_TIMEOUT=86400
export HF_HUB_DOWNLOAD_TIMEOUT=86400
export HF_ENDPOINT=https://john.jfrog.io/artifactory/api/huggingfaceml/john_HF_local
```

4. Authenticate the Hugging Face client with Artifactory by running the following command:

```
export HF_TOKEN=<IDENTITY_TOKEN>
```

For example:

```
export HF_TOKEN=ZtdKilLgbgprmklrUXyjD0uCGIHN7ZywZ55jCAwpHhTzsOr7Uv-L7oeCKZ3y
```

#### 7.19.3.1.1 | Configure Hugging Face SDK With Anonymous Access

Artifactory supports unauthenticated access to Hugging Face repositories.

To configure the Hugging Face client against Artifactory anonymously:

1. Log out of Hugging Face Hub on your local machine
  2. Add your repository using the following command:
- ```
export HF_HUB_ETAG_TIMEOUT=86400
export HF_ENDPOINT=<JFROG_HOST_URL>/artifactory/api/huggingfaceml/<REPOSITORY_KEY>
```

Make sure **not** to authenticate the Hugging Face client by passing your Hugging Face token.

### 7.19.3.2 | Deploy Hugging Face Models and Datasets

#### Warning

For Artifactory versions earlier than 7.75.3, it is only possible to deploy lighter models to Hugging Face local repositories due to limitations in the implementation of the Hugging Face client.

This topic describes how to deploy Hugging Face models and datasets to an Artifactory repository. Artifactory supports resolving datasets starting from version 7.90.x.

To deploy models or datasets to Artifactory using the `huggingface_hub` library insert your folder name, name, and revision number into the following command and run it:

#### Note

Make sure to replace the placeholders in angle brackets (<>) with your own model/dataset source folder name, name, and unique identifier.

#### Warning

Deploying a model or dataset using the same UUID as a cached model/dataset will cause the new one to override and replace the previous one.

```
from huggingface_hub import HfApi
api = HfApi()
api.upload_folder(
    folder_path="<MODEL/DATASET_SOURCE_FOLDER_NAME>",           # Enter the name of the folder on your local machine where your model resides.
    repo_id="<MODEL/DATASET_NAME>",       # Enter the name of the model, following the Hugging Face repository naming structure 'organization/name'. (models--$<MODEL_NAME>
    revision="<MODEL/DATASET_UUID>",           # Enter a unique identifier for your model. We recommend using a git commit revision ID.
    snapshots/$<REVISION_NAME>/...files)
```

# JFrog Artifactory Documentation

## Displayed in the header

```
repo_type="model"/"dataset"
)

For example:

from huggingface_hub import HfApi
api = HfApi()
api.upload_folder(
    folder_path="vit-base-patch16-224",           # Enter the name of the folder on your local machine where your model resides.folder to upload location on the FS
    repo_id="nsl319/legalpegasus",                 # Enter the name of the model, following the Hugging Face repository naming structure 'organization/name'. (models--$<MODEL_NAME>
    revision="54ef2872d33bbff28eb09544bdec6bf6699f5b0b8",          # Enter a unique identifier for your model. We recommend using a Git commit revision ID.
    snapshots/${<REVISION_NAME>}/...files)
    repo_type="model"
)
```

### 7.19.3.3 | Resolve Hugging Face Models and Datasets

This topic describes how to resolve Hugging Face models and datasets from within Artifactory. Artifactory supports resolving datasets starting from version 7.90.x.

To resolve a model from Artifactory, run the following command:

#### Note

Resolving models from Artifactory might take a while due to the size of the models.

#### Note

Make sure to replace the placeholder in angle brackets (<>) with your own model/ dataset name and revision number.

```
from huggingface_hub import snapshot_download
snapshot_download(
    repo_id=<MODEL_NAME>, revision=<MODEL_UUID>
)
```

For example:

```
from huggingface_hub import snapshot_download
snapshot_download(
    repo_id="nsl319/legalpegasus", revision="54ef2872d33bbff28eb09544bdec6bf6699f5b0b8"
)
```

To resolve a dataset from Artifactory, run the following command:

#### Note

Make sure to replace the placeholder in angle brackets (<>) with your own model/ dataset name and revision number.

```
from huggingface_hub import snapshot_download
snapshot_download(
    repo_id=<DATASET_NAME>, revision=<DATASET_UUID>, repo_type="dataset"
)
```

To find the revision ID for a model or dataset on Hugging Face Hub:

1. In the model or dataset page, go to the **Files and versions** tab.
2. Click the **History** button on the top right-hand side. This page contains the commit history of the model or dataset, each with its own Git commit revision ID.
3. Click the **copy** icon to copy the full commit hash to your clipboard.

### 7.19.3.3.1 | Resolve Hugging Face Models From Private Repositories

Starting from Artifactory version 7.77.x, Artifactory supports resolving models from private Hugging Face repositories.

To configure a remote repository to resolve models from a private repository:

1. Go to Hugging Face Hub, click on your profile on the top right side of the screen, and click **Settings**
2. Select **Access Tokens** from the sidebar menu
3. Select the token you'd like to use and click the **copy** icon to copy the token
  - a. (Optional) If you do not have a token, click **New token**, enter a name for the token and select the **read** role, and then click **Generate a token**

4. Go to your remote repository settings in Artifactory, and enter the token you copied from Hugging face in the **Password / Access Token** field. There is no need to fill in the **User Name** field. You can now resolve models from your private repository to your Hugging Face remote repository.

#### 7.19.3.3.2 | [Resolve Hugging Face Models and Datasets using Libraries](#)

Artifactory supports resolving specific models, datasets, and related files using libraries like Transformers and Diffusers, among others.

##### Note

Resolving Hugging Face models using libraries is only supported from Artifactory version 7.77 and above, Hugging Face client version 0.19.0 and above, and with the HF\_HUB\_ETAG\_TIMEOUT parameter enabled. Otherwise, use snapshot download to resolve whole model repositories.

##### Model Example

In the following example, we will use the Transformers library to download a model called Llama-2-7b-chat-hf using artifactory. Transformers is the library supported by this model: note that other models might have different library options.

To resolve the Llama-2-7b-chat-hf model from an Artifactory repository using Transformers:

1. Go to the model page on Hugging Face Hub, click **Use this model** on the top right side of the screen, and select **Transformers** from the drop-down menu

2. Click **Copy** next to the command you want to use

3. Run the command in your code to resolve the model

#### Dataset Example

In the following example, we will use the Datasets library to download a dataset called wikipedia using artifactory. Note that Datasets is the only library used by Artifactory to download datasets.

To resolve the wikipedia library from an Artifactory repository using Datasets:

1. Go to the dataset page on Hugging Face Hub, click **</> Use this dataset** on the top right side of the screen, and select **Datasets** from the drop-down menu

2. Click **Copy** next to the command you want to use

3. Run the command in your code to resolve the dataset.

## 7.20 | Kubernetes Helm Chart Repositories

### Helm Client Relative URL Support as Default (Helm Client V2 End of Life)

From November 1st, 2024, Artifactory will use [Helm Relative URL](#) for indexing Helm Chart responses by default.

Since this feature is only supported by Helm clients V3 and above, customers using Helm clients V2 are required to upgrade.

From January 1st, 2025, Artifactory will use Helm client v3 by default, and remove the support for all Helm client versions under 3.0.0.

For more information, see [Deprecations in Process](#).

Artifactory offers fully-featured operation with Helm through support for local, remote, and virtual Helm chart repositories.

Artifactory's support for Helm charts includes:

- Secure, private repositories for Helm charts with fine-grained access control according to projects or development teams.
- Calculation of metadata for Helm charts hosted in Artifactory local repositories.
- Access to remote Helm chart repositories through [remote repositories](#) which provide proxy and caching functionality.
- Support for [Helm OCI repositories](#), enabling you to use OCI to store and share Helm charts.
- Enterprise features such as high availability, repository replication for multi-site development and different options for massively scalability storage.
- Supports Helm 3 clients, enabling you to deploy and resolve Helm Charts using Helm V2 and V3 clients.

### Learn More

[K8S cheatsheet for managing applications](#)

### 7.20.1 | Helm OCI Repositories

Helm has adopted OCI as the primary solution for scale and distribution as of Helm version 3.8.0. This overcomes scaling issues with the `index.yaml` file, making it easier to scale, distribute, and leverage the power of OCI to deliver charts.

Starting from Artifactory version 7.75.3, Artifactory defaults repository creation to Helm OCI in the JFrog Platform WebUI and provides legacy support for older versions.

#### 7.20.1.1 | Set Up a Helm OCI Repository

You can create several types of Helm OCI Repositories using the JFrog Platform WebUI:

- Local Helm OCI Repositories
- Remote Helm OCI Repositories
- Virtual Helm OCI Repositories

To set up a Helm OCI repository via REST API, use the Create Repository REST API and add the parameter "packageType" : "helmoci" to the Repository Configuration JSON file.

Starting from Artifactory version 7.75.3, when creating a new Helm repository, a [Helm OCI](#) repository will be created by default that is compliant with OCI, and supports OCI REST APIs instead of Helm.

You can change this setting in two ways:

- To use the legacy Helm implementation for one repository, click the [Use Legacy Helm](#) button on the repository creation menu.
- To change the default behavior to Helm instead of Helm OCI, navigate to [Administration > Artifactory > Packages > UI settings](#), and deselect the [Helm repository UI : Show OCI Flow first](#) checkbox.

# JFrog Artifactory Documentation Displayed in the header

## 7.20.1.1.1 | Set Up Local Helm OCI Repositories

A local Helm OCI repository is where you deploy and host your internal Helm OCI resources. It is, in effect, a Helm OCI registry able to host collections of tagged Helm OCI resources which are your Helm OCI 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 create a local Helm OCI repository:

1. From the Administration module, select **Repositories | Repositories | Local**.
2. Click **New Local Repository** and select **Helm** from the **Select Package Type** dialog.
3. Set the **Repository Key**.
4. (Optional) Set **Max Unique Tags**. This specifies the maximum number of unique tags, per repository, that should be stored for a OCI 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. For more information, see [Use Max Unique Tags](#).
5. (Optional) Set **Tag Retention**. This specifies the number of tags that the JFrog Platform will retain when they are overwritten. Leaving the field at 1 (default) means that no overwritten tags will be saved. For more information, see [Use Tag retention](#).

## 7.20.1.1.2 | Set Up Remote Helm OCI Repositories

Artifactory supports proxying remote Helm OCI registries through remote repositories. A Remote Repository in Artifactory serves as a caching proxy for a registry managed at a remote URL such as Helm Registry.

Resources that are requested from a remote repository are cached on demand. You can remove downloaded resources from the remote repository cache, however, you can not manually push resources to a remote repository.

To create a remote repository to proxy a remote OCI registry, follow these steps:

1. From the Administration module, select **Repositories | Repositories | Remote**.
  2. Click **New Remote Repository** and select **Helm** from the **Select Package Type** dialog.
  3. In the Basic tab, set the **Repository Key** value, and specify the URL to the remote registry in the **URL** field.
  4. To allow Artifactory to download foreign layers to the Helm OCI remote repository, click the **Advanced** tab and select the **Enable Foreign Layers Caching** checkbox.
  5. (Optional) To match external URL when downloading from foreign layers, set include/exclude patterns. These are Ant-style expressions that allow you to specify where foreign layers may be downloaded from: supported expressions include \*, \*\*, ?.
- For example, setting the pattern `**/github.com/**` will allow the downloading of foreign layers only from the `github.com` host.
- The default setting for this field is `**`, which allows the download of foreign layers from any source.
6. Configure the network settings. For more information, see [Network Settings](#).
  7. Click **Save and Finish**.

## Smart Remote Repository Path and Domain

When accessing a Smart Remote OCI repository through Artifactory, the repository URL must be prefixed with `api/oci` in the path:

### Tip

Make sure to replace the placeholder with your JFrog domain information.

`<YOUR_JFROG_DOMAIN>/artifactory/api/oci/<REPOSITORY>`

For example:

`https://my-awesome-jfrog.io:8081/artifactory/api/oci/jfrog-oci-remote`

## 7.20.1.1.3 | Set Up Virtual Helm OCI Repositories

Artifactory supports virtual Helm OCI Repositories. A **Virtual Repository** 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 Helm OCI repositories, as well as remote images that are proxied by remote Helm OCI repositories, and access all of them from a single URL defined for the virtual repository.

Virtual repositories can be very useful since users will continue to work with the virtual repository while the admin can manage the included repositories, and replace the default deployment target, while the changes will be transparent to the users.

To create a virtual Helm OCI repository:

1. From the Administration module, select **Repositories | Repositories | Virtual**.
2. Click **New Virtual Repository** and select **Helm** from the **Select Package Type** dialog.
3. Set the **Repository Key** value.
4. Select the underlying local and remote Helm OCI repositories to include under the **Repositories** section.
5. (Optional) Configure your **Default Deployment Repository**, where the images you upload to this virtual repository will be routed. Once you configure this repository, your Helm OCI repository will be fully fledged.

## Use Virtual Repositories to Represent Your Pipeline

You can also set up your virtual repository to wrap a series of repositories representing the stages of your development pipeline, and then use them to promote resources from the default deployment repository through the pipeline to production.

# JFrog Artifactory Documentation

## Displayed in the header

All of the repositories representing pipeline stages can also be configured with different access permissions, to tailor the pipeline to your needs.

### 7.20.1.2 | Set Up Helm OCI Client To Work With Artifactory

This section describes how to set up Helm OCI to work with Artifactory. It includes the following guides:

- [Configure the Helm OCI Client](#)
- [Push Helm OCI Charts](#)
- [Pull Helm OCI Charts](#)

#### 7.20.1.2.1 | Configure the Helm OCI Client

##### Important

These instructions are relevant for Helm OCI Repositories and intended for use with Helm client version 3.8.0 and later (or for prior versions with OCI Experimental mode enabled). Helm OCI repositories do not support non-OCI interactions. For more information, see the [Helm documentation](#).

To configure the Helm OCI client with Artifactory, log in using the `helm registry login` command:

##### Note

Make sure to replace the placeholder in **bold** with your own JFrog host domain.

```
helm registry login <YOUR_JFROG_DOMAIN>
```

Alternatively, you can provide your username and token in advance using the following command:

##### Note

Make sure to replace the placeholders in **bold** with your own username, token, and JFrog host domain.

```
helm registry login -u <USER_NAME> -p <TOKEN> <YOUR_JFROG_DOMAIN>
```

### Use Helm OCI With Older Client Versions

Starting from Artifactory version 7.75.3, To use Helm OCI with a Helm client version prior to 3.8.0, run this command in your Helm client:

```
export HELM_EXPERIMENTAL_OCI=1
```

For more information, see the [Helm documentation](#).

### Use an encrypted password

We recommend using an encrypted password instead of clear-text. For details, please refer to [Centrally Secure Passwords](#).

#### 7.20.1.2.2 | Push Helm OCI Charts

To push a chart in OCI form, use the `helm push` command:

##### Important

Make sure you add the `oci://` prefix to the Artifactory repository URL to force the Helm chart to wrap the artifact as OCI.

##### Note

Make sure to replace the placeholders in **bold** with your Helm chart .TGZ file, URL, and desired repository path.

```
helm push <HELM_CHART_TGZ_FILE> oci://<URL>/<REPOSITORY>
```

#### 7.20.1.2.3 | Pull Helm OCI Charts

You can pull a Helm chart using the following methods:

- [Pull specific version](#)
- [Pull latest version](#)

##### Pull Specific Version

To pull a specific chart, use the following command:

##### Note

Make sure to replace the placeholders in **bold** with your own JFrog domain, repository name, chart name, and chart version.

```
helm pull oci://<YOUR_JFROG_DOMAIN>/<REPOSITORY_NAME>/<CHART_NAME> --version <CHART_VERSION>
```

##### Pull Latest Version

To pull the latest chart version, use the following command:

##### Note

Make sure to replace the placeholders in **bold** with your own JFrog domain, repository name, and chart name.

# JFrog Artifactory Documentation

## Displayed in the header

helm pull oci://<YOUR\_JFROG\_DOMAIN>/<REPOSITORY\_NAME>/<CHART\_NAME>

### 7.20.1.3 | [View Individual Helm OCI Artifact Information](#)

Artifactory displays selected metadata of a Helm OCI artifact in the JFrog Platform WebUI.

In the Tree Browsing tab under the Artifacts screen, select your virtual Helm OCI repository and drill down to find and select the package you want to inspect. The metadata is displayed in the General and Info tabs.

Starting from Artifactory version 7.75, in the Packages view, both Helm OCI and Helm will show up side by side, but in the Artifacts view, you'll be able to filter and view only one or the other.

Starting from Artifactory version 7.90.1, you can use Referrers API to link between images and their related information. For more information, see [Use Referrers REST API to Discover OCI References](#).

### 7.20.1.4 | [Limitations of Helm OCI](#)

- Artifactory does not support adding both types of Helm repositories (Helm and Helm OCI) in the same virtual repository. You can have two virtual repositories, one for Helm and one for Helm OCI.
- Xray does not support scanning Helm charts.
- Index manifests are not automatically moved, copied, or deleted with their related images. When performing these actions, select both the index manifests and the related images.
- Since OCI can support multiple formats, Helm OCI is not limited to storing just Helm charts: although it is described as Helm, is compliant with OCI and allows you to upload any file type.

### 7.20.2 | [Helm Repositories](#)

Artifactory's support for Helm charts includes:

- Calculation of metadata for Helm charts hosted in Artifactory local repositories.

#### 7.20.2.1 | [Create Helm Repositories](#)

You can create several types of Helm Repositories:

- Local Helm Repositories
- Remote Helm Repositories
- Virtual Helm Repositories

##### 7.20.2.1.1 | [Set Up Local Helm Repositories](#)

To enable the calculation of Helm chart metadata, from the **Administration** module, select **Repositories** | **Repositories** | **Local** and set **Helm** to be the **Package Type** when you create your local repository.

7.20.2.1.2 | Set Up Remote Helm Repositories

You can create Helm remote repositories to proxy and cache remote repositories or other Artifactory instances.

In order for Artifactory to properly cache Helm charts, resolve the charts only through a virtual repository.

7.20.2.1.2.1 | Automatically Rewrite External Dependencies for Helm Charts

Helm Charts requested by the Helm client frequently use external dependencies as defined in the `index.yaml` file. These dependencies may, in turn, need additional dependencies. Therefore, when downloading a chart, you may not have full visibility into the full set of dependencies that your original chart 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 charts through Artifactory, you may specify a "safe" Allow List from which dependencies may be downloaded, cached in Artifactory, and configured to rewrite the dependencies so that the Helm client accesses dependencies through a remote repository as follows:

- Select the **Enable Dependency Rewrite** checkbox in the Helm Chart remote repository advanced section.
- Specify an Allow List pattern of external resources from which dependencies may be downloaded.

The fields under **External Dependency Rewrite** are connected to automatically rewriting external dependencies for Helm Charts that require them.

| Field                     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Enable Dependency Rewrite | When selected, external dependencies are rewritten.                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Patterns Allow List       | An Allow 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.<br><br>For example, if you limit the Patterns Allow List to <code>https://github.com/**</code> , the external dependencies will be cached in the "helm" remote repository, and only charts with a URL starting with <code>https://github.com/</code> will be allowed to be cached. |

For example, if you limit the Patterns Allow List to "github.com"github.com, the external dependencies will be cached in the "helm" remote repository, and only charts from `https://github.com/prometheus-community/helm-charts/` are allowed to be cached.

7.20.2.1.3 | Set Up Virtual Helm Repositories

A Virtual Repository defined in Artifactory aggregates packages from both local and remote repositories.

This allows you to access both locally hosted Helm charts and remote proxied Helm charts repositories from a single URL defined for the virtual repository.

To define a virtual Helm chart repository, create a **virtual repository**, set the **Package Type** to be **Helm**, and select the underlying local and remote Helm repositories to include in the **Basic** settings tab.

This repository will be configured in the Helm client.

7.20.2.1.3.1 | Namespace Support for Helm Virtual Repositories

# JFrog Artifactory Documentation Displayed in the header

From Artifactory 7.24.1 (SaaS Version), you can explicitly state a specific aggregated local or remote repository to fetch from a virtual by assigning namespaces to local and remote repositories in Helm virtual repositories according to the following syntax.

By default, this feature is disabled but can be set when creating or updating a virtual Helm repository.

```
/helm-virtual/<local_repository_name>/chart.tgz
```

7.20.2.1.3.2 | [Relative URL Support for Helm Repositories](#)

From version 7.59.5, Artifactory supports relative URLs for indexing Helm charts. Relative URLs allow the Helm client to use shorter URLs, which improves the performance of your Helm client and slightly reduces the size of your `index.yaml` file. This feature is available for Helm clients in version 3 and up.

To use this feature, enable the following feature flag in your system configuration file:

```
artifactory.helm.relative.urls.enabled=true
```

This will only apply to new chart deployments- if you want it to apply to all charts, run a simple reindex to align the index.

7.20.2.1.3.3 | [Set Multiple External Dependencies for Helm Using a List of URLs](#)

To support downloading files from multiple internal Artifactory URLs, you can create a list of URLs that are trusted by the repository.

By default, this feature is disabled but can be set for each aggregated remote repository separately.

The following example shows how to add the following configuration as an external dependency.

```
https://example.com/example-community/helm-charts/**
```

As a result, all the external URLs located in the `index.yaml` file starting with the following pattern.

```
https://example.com/example-community/helm-charts/
```

Will be replaced with the following syntax,

```
http://rt-host/artifactory/api/helm/helm-virtual/_external/https://example.com/example-community/helm-charts/
```

7.20.2.1.3.4 | [Helm Virtual Repository Index Improvements](#)

Starting from Artifactory version 7.80.0, we have improved our index calculation mechanism for virtual repositories to minimize potential OOM issues.

## Important

If you use a Helm virtual repository with a virtual cache expiration time set to 0, you might experience slower resolution times.

We recommend setting the **Metadata Retrieval Cache Period (Sec)** in the repository page in the JFrog Platform WebUI to 60 seconds or more. This will help you avoid recreating the index every time there's a new request and prevent out-of-memory issues. By default, this parameter is set to 6 minutes. This will prevent subsequent indexing requests from being triggered during this time period.

When receiving a download request for an `index.yaml` file against a Helm virtual repository, Artifactory will perform the following steps:

1. Check whether the instance already contains a cached non-expired calculated file that fits the user's permissions, and return that file if it exists.
2. If the file does not already exist, check if there is an ongoing calculation of the merged `index.yaml` file that fits the user's permissions. If the calculation is in progress, wait 6 minutes or until the calculation is completed, whichever happens first.
3. If the calculation is completed within 6 minutes, whether the instance contains a cached non-expired calculated file that fits the user's permissions, and return that file if it exists.

You can use this feature flag to return to the previous behavior:

```
artifactory.virtual.repo.metadata.merge.lock.timeout.sec.HELML=120
```

7.20.2.2 | [Set Up Kubernetes Helm Charts To Work With Artifactory](#)

This section describes how to set up Helm to work with Artifactory. It includes the following guides:

- Configure the Helm Client
- Deploy Helm Charts
- Resolve Helm Charts

7.20.2.2.1 | [Configure the Helm Client](#)

## Download Helm client version 2.9.0 or above for authenticated access

To use all features of Artifactory Helm chart repositories, including resolution of Helm charts, you must use version 2.9.0 or above of the Helm client that supports basic authenticated access to Artifactory.

Before you can use your Helm client to resolve Helm charts from Artifactory, you need to configure it for authenticated access with your Artifactory user and password by adding the virtual Helm chart repository to be used for resolution as shown below:

```
helm repo add <REPO_KEY> http://<ARTIFACTORY_HOST>:<ARTIFACTORY_PORT>/artifactory/<REPO_KEY> --username <USERNAME> --password <PASSWORD>  
helm repo update
```

For example:

# JFrog Artifactory Documentation Displayed in the header

```
helm repo add helm-virtual http://10.1.16.114:32775/artifactory/helm-virtual --username admin --password password
helm repo update
```

## Use an encrypted password

We recommend using an encrypted password instead of clear-text. For details, please refer to Centrally Secure Passwords.

7.20.2.2.1.1 | [Use the JFrog Helm Client](#)

## JFrog Helm Client

The JFrog Helm Client was necessary for authenticated access to Artifactory before the Helm client supported basic authentication (before version 2.9.0).

If you are using the JFrog Helm Client, you need to configure it for authenticated access with your Artifactory user and password by adding the virtual Helm chart repository to be used for resolution as shown below:

```
helm repo add <REPO_KEY> http://<ARTIFACTORY_HOST>:<ARTIFACTORY_PORT>/artifactory/<REPO_KEY> <USERNAME> <PASSWORD>
helm repo update
```

For example:

```
helm repo add helm-virtual http://10.1.16.114:32775/artifactory/helm-virtual admin password
helm repo update
```

7.20.2.2.2 | [Deploy Helm Charts](#)

Deploying Helm charts is done using cURL, Wget, JFrog CLI or any of the ways described in [Deploying Artifacts](#).

To deploy Helm charts to a virtual Helm repository, make sure you have set the **Default Deployment Repository**.

## Reindexing a Helm Chart repository

You can trigger an asynchronous reindexing of a local Helm chart repository either through the UI or using the REST API.

Through the UI, select your Helm chart repository in the Tree Browser and select **Recalculate Index** from the right-click menu, as shown below (requires Admin privileges).

To reindex a Helm chart repository through the REST API, please refer to [Calculate Helm Chart Index](#).

## Warning

Using the above REST API command or UI will reindex the `index.yaml` from scratch. You may receive a partial `index.yaml` if you attempt to resolve the package from the repository while the calculation is ongoing. Therefore, reindex should only be used if the `index.yaml` is corrupted.

7.20.2.2.3 | [Resolve Helm Charts](#)

JFrog Artifactory supports the resolution of Helm charts from local and virtual Helm chart repositories. To resolve Helm charts from remote Helm chart repositories, you need to aggregate them in a virtual Helm chart repository.

To resolve a Helm chart through Artifactory, use the following command:

```
helm install <REPO_KEY>/<CHART_NAME>
```

For example:

# JFrog Artifactory Documentation

## Displayed in the header

```
helm install helm-virtual/artifactory
```

### 7.20.2.3 | View Individual Helm Chart Information

Artifactory displays selected metadata of a Helm chart in its UI. Artifactory lets you view selected metadata of a Helm chart directly from the UI.

In the **Tree Browsing** tab, select your virtual Helm chart repository and drill down to find and select the package you want to inspect. The metadata is displayed in the **Chart Info** tab.



### 7.20.2.4 | Helm Charts Partial Re-Indexing

Artifactory supports partial re-indexing of Helm charts. This allows users to manually trigger the re-indexing process to address issues with charts that are missing from (or corrupted in) the **index.yaml** file. Partial re-indexing reduces processing time and resource usage by enabling more efficient and targeted updates. Partial re-indexing can be triggered on a path containing multiple charts to be re-indexed, as well as on a specific chart archive. Click here to see documentation on the Helm Charts Partial Re-Indexing REST API.

#### Note

Helm Charts Partial Re-Indexing is available starting from Artifactory version 7.105.2.

### 7.20.2.5 | Helm Enforce Layout

Helm Enforce Layout is designed to maintain the integrity and organization of Helm charts within your repositories. It consists of two key functionalities that promote structure and reduce errors during deployments:

- Preventing duplicate chart paths
- Enforcing chart names and versions

#### Note

Helm Enforce Layout is forward-compatible only, it will not work on repositories created prior to Artifactory 7.104.2. This means that even if you upgrade to Artifactory 7.104.2, any repositories created prior to the upgrade are not compatible with this feature. Enforcement is set only upon repository creation.

#### 7.20.2.5.1 | Prevent Duplicate Chart Paths

This functionality prevents the deployment of charts with the same name and version to different paths within the same repository, by ensuring that only a single instance of a chart is indexed. This maintains the integrity and accessibility of Helm charts, ensuring that users can easily identify and resolve the desired version without confusion.

#### Example

When deploying a chart named **chart-1.0.0.tgz** where there is already **chart-1.0.0.tgz** in the repository under a different path, Artifactory will reject the upload with a 403 response code and proper message.

This action is prevented due to the Enforce Layout Policy, a package with the same name and version **chart-1.0.0** already exists in the repository.

#### 7.20.2.5.2 | Enforce Chart Name and Version

This functionality ensures that the chart name and version specified in the packaged file name match the values in **Chart.yaml** and adhere to Semantic Versioning (SemVer2) standards adopted by the Helm official specification. Enforcing these rules promotes uniformity, allowing teams to adopt clear naming conventions that foster better collaboration and understanding of changes across different versions.

#### Example 1:

When deploying a chart with non-SemVer2 version (1.0.0.0 for example), Artifactory will reject the upload with a 403 response and a proper message.

This action is prevented due to the Enforce Layout Policy, the specified package version **chart-1.0.0.0** does not comply with the Semantic Versioning format.

#### Example 2:

When deploying a chart under **chart-1.0.0.tgz**, the chart name and chart version inside the **Chart.yaml** file must be **chart** and **1.0.0** respectively, otherwise Artifactory will reject the upload with a 403 response code and a proper message.

# JFrog Artifactory Documentation

## Displayed in the header

This action is prevented due to the Enforce Layout Policy, the package file `chart-1.0.` does not match the package name/version `chart:2.0.0` in the metadata file.

### Example 3:

When deploying a chart with a malformed name or version, Artifactory will reject the upload with a 403 response code and a proper message.

This action is prevented due to the Enforce Layout Policy, the metadata of the package `chart-1.0.0` could not be read or is malformed.

7.20.2.5.3 | [Activate Helm Enforce Layout](#)

Enforcement can be enabled either globally for all Helm repositories in Artifactory, or for specific Helm repositories. If you enable enforcement globally, the enforcement cannot be later disabled on any Helm repositories in Artifactory.

### Enable Enforcement Globally

To enable global enforcement, in Artifactory click the Administration tab and go to [Artifactory Settings->Packages Settings](#). The Package Settings screen appears, as shown below.

In the Package Settings screen, select the check boxes by **Prevent Duplicate Chart Paths** and **Enforce Chart Name and Version**, then click **Save**.

### Enable Enforcement on Specific Repositories

System Administrators can selectively activate enforcement on specific repositories. This allows tailored enforcement that fits the specific needs of different projects or teams.

#### To enable enforcement on a specific repository:

1. In Artifactory click the **Administration** tab and click **Repositories** in the navigation panel on the left. The Repositories screen appears, as shown below.

2. In the Repositories screen, click the **Create a Repository** button and then **Local** from the drop-down menu. The Select Package Type window appears, as shown below.

3. In the Select Package Type window, select **Helm**. The New Local Repository screen appears, as shown below.

4. In the New Local Repository screen, click **Use Legacy Helm**. At the bottom-right of the screen, the Enforcement Settings appear, as shown below.

5. Under Enforcement Settings, select the check boxes by **Prevent Duplicate Chart Paths** and **Enforce Chart Name and Version**, then click **Create Local Repository**.

### Note

When two or more local repositories under a virtual repository have the same chart name and version, the resolution will be according to the repositories' [resolution order](#).

7.20.2.5.4 | [Limitations on Helm Enforce Layout](#)

- **Push Replication/Copy/Move:** If the source is not set with enforcement, and the target is set with enforcement, it is possible that not all charts will be replicated/copied/moved successfully because the target repository might filter charts that are breaching enforcement.

7.20.2.6 | [Watch the Helm Screencast](#)



## 7.21 | Machine Learning Repositories

Machine Learning Repositories with the FrogML SDK is a local management framework tailored for machine learning projects, serving as a central storage for models and artifacts, featuring a robust version control system. It offers local repositories and an SDK for effortless model deployment and resolution.

### Note

Machine Learning Repositories is available on Artifactory Self-Hosted platforms starting from Artifactory Self-Hosted 7.104.4.

### 7.21.1 | Main Features of Machine Learning Repositories in Artifactory

- **Secure Storage:**  
Protect your proprietary information by deploying models and additional resources to Artifactory local repositories, giving you fine-grain control of the access to your models.
- **Easy Collaboration:**  
Share and manage your machine learning projects with your team efficiently.
- **Easy Version Control:**  
The Machine Learning Repositories SDK (FrogML) provides a user-friendly system to track changes to your projects. You can name, categorize (using namespaces), and keep track of different versions of your work.

### 7.21.2 | Machine Learning Repositories Configuration Overview

Working with Machine Learning Repositories includes the following main steps:

| Name                                 | Description                                                                                                                                                                | For more information, see:                           |
|--------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------|
| Create Machine Learning Repositories | Learn how to create local Machine Learning Repositories.                                                                                                                   | <a href="#">Create Machine Learning Repositories</a> |
| FrogML Library                       | Learn how to configure the FrogML client opposite Artifactory, upload models to the Machine Learning repository, and download models from the Machine Learning repository. | <a href="#">FrogML Library</a>                       |

For information about Machine Learning and FrogML limitations, see [Limitations of Machine Learning Repositories in Artifactory](#).

### 7.21.3 | Create Machine Learning Repositories

To create a local Machine Learning repository:

1. From the **Administration** module, go to the **Repositories** page and click **Add Repositories**.
2. Select **Local Repository** from the drop-down menu and select **Machine Learning** from the Select Package Type dialog.

3. Set the **Repository Key**.

#### 7.21.4 | Machine Learning Repository Structure

The FrogML repository structure for local repositories is as follows:

```
└─ ${REPOSITORY_NAME}
    └─ models
        └─ ${NAMESPACE}
            └─ ${MODEL_NAME}
                └─ ${MODEL_VERSION}
                    └─ model-manifest.json
                    └─ model
                        └─ file1
                        └─ file2
                    └─ code
                        └─ code.zip
                        └─ requirements.txt
```

#### 7.21.5 | Limitations of Machine Learning Repositories in Artifactory

Machine Learning Repositories in Artifactory have the following limitation:

- Currently, Xray scans on Machine Learning resources are not supported.

#### 7.21.6 | FrogML Library

The FrogML library is a powerful new Python tool designed to streamline the Machine Learning (ML) models and associated artifacts stored in JFrog Artifactory. With FrogML, you can upload and download models seamlessly and efficiently.

The FrogML library currently contains a **file-based model type**, and **6 format-aware model types**. The format-aware model types are:

- CatBoost
- Hugging Face
- ONNX
- scikit-learn
- Generic Python Function
- PyTorch

All model types contain the following 3 functions:

- Upload a model to a Machine Learning repository
- Download a model from a Machine Learning repository
- Get model information from a Machine Learning repository

# JFrog Artifactory Documentation

## Displayed in the header

### Install FrogML Client

To use Machine Learning Repositories in Artifactory, you need to download the FrogML SDK library client. To get access to the FrogML client, install it with the following command:

```
pip install frogml
```

### Authenticate FrogML with Environment Variables

To authenticate the FrogML with Artifactory define the following environment variables:

- **JF\_URL**: Your JFrog platform domain (e.g., <http://myorg.jfrog.io>)
- **JF\_ACCESS\_TOKEN**: Your Artifactory token for this domain. To generate a token, log in to Artifactory, navigate to your FrogML repository, and click on **Set Me Up**.

Once these environment variables are set, you can start using FrogML.

#### 7.21.6.1 | File-Based Model Type

Use the file-based model type to upload, download, or get information on files not supported by one of the format-aware file types in the FrogML SDK.

### Upload a File-Based Model to a Machine Learning Repository

You can upload a model to a Machine Learning repository using the `frogml.log_model()` function. A model can be a single file or a list of files in a directory. This function uses checksum upload, assigning a SHA2 value to each model for retrieval from storage. If the checksum fails, FrogML implements regular upload.

After uploading the model, FrogML generates a file named `model-manifest.json`, which contains the model name and its related files and dependencies.

To upload a file-based model to a Machine Learning repository, use the following function:

```
import frogml
frogml.files.log_model(repository=<REPOSITORY_KEY>,      # The JFrog repository to upload the model to.
                       namespace=<NAMESPACE_NAME>,    # Optional. The namespace or organization.
                       model_name=<MODEL_NAME>,       # The uploaded model name
                       version=<MODEL_VERSION>,       # Optional. The uploaded model version
                       source_path=<FILE_PATH>,        # The file path of the model on your machine.
                       properties=<PROPERTIES>,        # Optional. These are properties you can set on the model to categorize and label it.
                       dependencies=[<DEPENDENCY>, <DEPENDENCY>], # Optional
                       code_path= <CODE_PATH>
)
```

### Note

Make sure to replace the placeholders in bold with your own JFrog repository key, namespace name, model name, source path, version, properties, dependencies, and code path. If you do not specify a version, the version will be saved as the timestamp in UTC format.

### Parameters

| Parameter        | Description                                                                                                                                                                                                                                                                                                                   | Example                                                                                 |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
| <REPOSITORY_KEY> | The name of the JFrog repository you want to upload the model to                                                                                                                                                                                                                                                              | frogml-local                                                                            |
| <NAMESPACE_NAME> | (Optional) The name of the namespace or organization: use this parameter to group models by project or group.                                                                                                                                                                                                                 | frog-ml-beta                                                                            |
| <MODEL_NAME>     | The name of the model you want to upload                                                                                                                                                                                                                                                                                      | my-cool-model                                                                           |
| <MODEL_VERSION>  | (Optional) The version of the model you want to upload.<br>If you do not add a model version, Artifactory will set the version as the timestamp of the time you uploaded the model in your time zone, in UTC format: yyyy-MM-dd-HH-mm-ss.                                                                                     | 1.0.0                                                                                   |
| <FILE_PATH>      | The file path of the model on your machine or to the directory you want to upload.                                                                                                                                                                                                                                            | /root/models/my-cool-model                                                              |
| <PROPERTIES>     | These are properties you can set on the model to categorize and label it, in the format: "x": "1", "y": "2"                                                                                                                                                                                                                   | "model_type": "keras", "experiment": "my-exp"                                           |
| <DEPENDENCIES>   | A list of the package types and their corresponding versions that your project is dependent on. The following dependency types are supported: <ol style="list-style-type: none"><li>1. requirements.txt for pip</li><li>2. poetry requirements file</li><li>3. conda requirements file</li><li>4. Explicit versions</li></ol> | For explicit versions:<br><code>dependencies = ["pandas==1.2.3", "numpy==1.2.3"]</code> |

| Parameter   | Description                                                                                                                                                                      | Example |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
|             | <b>Note</b><br><br>When providing a path to a requirements file, please provide the path as a parameter in a list.                                                               |         |
| <CODE_PATH> | An optional path to the code directory. When provided, the code files in the provided path will be uploaded to the Machine Learning repository together with the model artifact. | ./src   |

#### Example

```
import frogml
frogml.files.log_model(
repository="frog-ml-local",
namespace="frog-ml-beta",      # Optional
model_name="my-cool-model",
version="1.0.0",               # Optional
source_path "~/root/models/my-cool-model/",
properties={"model_type": "keras", "experiment": "my-exp"} # Optional
dependencies=["numpy>=1.19"], # Optional
code_path= "./src" #Optional
)
```

#### Download a File-Based Model from a Machine Learning Repository

You can use the `frogml.load_model()` function to download a file-based model from an Artifactory repository. This method will download the model file locally and return the path to the returned model.

```
import frogml

# Download model version
frogml.files.load_model(
    repository="<REPOSITORY_KEY>", # The name of the JFrog repository you want to download the model from
    namespace="<NAMESPACE_NAME>", # Optional. The name of the namespace or organization: use this parameter to group models by project or group.
    model_name="<MODEL_NAME>", # The name of the model you want to download.
    version="<MODEL_VERSION>", # The version of the model you want to download.
    target_path="<FILE_PATH>", # The file path to where you want to download the model.
)
```

#### Note

Make sure to replace the placeholders in bold with your own JFrog repository key, namespace name, model name, target path, and version.

#### Parameters

| Parameter        | Description                                                                                                   | Example       |
|------------------|---------------------------------------------------------------------------------------------------------------|---------------|
| <REPOSITORY_KEY> | The name of the JFrog repository you want to download the model from.                                         | frogml-local  |
| <NAMESPACE_NAME> | (Optional) The name of the namespace or organization: use this parameter to group models by project or group. | frog-ml-beta  |
| <MODEL_NAME>     | The name of the model you want to download.                                                                   | my-cool-model |
| <MODEL_VERSION>  | The version of the model you want to download.                                                                | 1.0.0         |
| <TARGET_PATH>    | The file path to where you want to download the model.                                                        | /root/models/ |

#### Example

```
import frogml

# Download model version
frogml.files.load_model(
    repository="frog-ml-local",
    namespace="frog-ml-beta",      #optional
    model_name="my-cool-model",
    version="1.0.0"
    target_path "~/root/models/",
)
```

# JFrog Artifactory Documentation

## Displayed in the header

### Return Value

The `load_model()` method returns the path to the local downloaded model as a `Path` object imported from `pathlib`.

### Get Information on a File-Based Model in a Machine Learning Repository

You can use the `frogml.get_model_info()` function to retrieve information on a specific file-based model version without downloading the full model files.

```
import frogml

# Download model version
frogml.files.get_model_info(
    repository="<REPOSITORY_KEY>", # The name of the JFrog repository you want to download the model from
    namespace="<NAMESPACE_NAME>", # Optional. The name of the namespace or organization: use this parameter to group models by project or group.
    model_name="<MODEL_NAME>", # The name of the model you want to download.
    version="<MODEL_VERSION>", # The version of the model you want to download.

)
```

### Note

Make sure to replace the placeholders in bold with your own JFrog repository key, namespace name, model name, and version.

### Parameters

| Parameter                     | Description                                                                                                   | Example       |
|-------------------------------|---------------------------------------------------------------------------------------------------------------|---------------|
| <b>&lt;REPOSITORY_KEY&gt;</b> | The name of the JFrog repository you want to download the model from.                                         | frogml-local  |
| <b>&lt;NAMESPACE_NAME&gt;</b> | (Optional) The name of the namespace or organization: use this parameter to group models by project or group. | frog-ml-beta  |
| <b>&lt;MODEL_NAME&gt;</b>     | The name of the model you want to download.                                                                   | my-cool-model |
| <b>&lt;MODEL_VERSION&gt;</b>  | The version of the model you want to download.                                                                | 1.0.0         |

### Example

```
import frogml

# Download model version
frogml.files.get_model_info(
    repository="frog-ml-local",
    namespace="frog-ml-beta",      #optional
    model_name="my-cool-model",
    version="1.0.0",

)
```

### Return Values

| Key                               | Description                                                                                                        | Example                                                                                                                                    |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <code>model_format</code>         | Details about the stored model such as framework, framework_version, runtime environment and serialization format. | <pre>{'framework': 'files', 'framework_version': '', 'runtime': 'python', 'runtime_version': '3.9.6', 'serialization_format': 'pk1'}</pre> |
| <code>model_artifacts</code>      | List of objects with details about the model artifact files.                                                       | <pre>[{"artifact_path": "&lt;artifactory_path&gt;", "checksum": "&lt;file_checksum&gt;", "download_path": "&lt;download_url&gt;}]</pre>    |
| <code>dependency_artifacts</code> | List of objects with details about the attached dependencies and requirement files.                                | <pre>[{"artifact_path": "&lt;artifactory_path&gt;", "checksum": "&lt;file_checksum&gt;", "download_path": "&lt;download_url&gt;}]</pre>    |
| <code>code_artifacts</code>       | List of objects with details about the attached code files                                                         | <pre>[{"artifact_path": "&lt;artifactory_path&gt;", "checksum": "&lt;file_checksum&gt;", "download_path": "&lt;download_url&gt;}]</pre>    |
| <code>created_date</code>         | Creation date in ISO 8601 format.                                                                                  | e.g. 2024-11-12T13:25:53.20                                                                                                                |

# JFrog Artifactory Documentation

## Displayed in the header

### 7.21.6.2 | Format-Aware Model Types

The format-aware FrogML SDK connects AI/ML frameworks with Artifactory as a single source of truth, enabling seamless integration and model management. This unification streamlines deployment, optimization, and governance, helping enterprises scale trusted AI applications effortlessly. Additionally, by working with the FrogML SDK, you gain access to experiment management and advanced ML capabilities using JFrog ML.

The FrogML library contains the following format-aware model types:

- CatBoost
- Hugging Face
- ONNX
- scikit-learn
- Generic Python Function
- PyTorch

#### 7.21.6.2.1 | CatBoost Model Type

##### Upload a CatBoost Model to a Machine Learning Repository

Use the following function to upload a CatBoost model to a Machine Learning repository in Artifactory:

```
import frogml

repository = "repository-name"
name = "model-name"
namespace = "namespace"
version = "version-1"
properties = {"key1": "value1"}
dependencies = ["path/to/dependencies/pyproject.toml", "path/to/dependencies/poetry.lock"]
code_dir = "full/path/to/code/dir"
catboost_model = get_catboost_model() # Function that Returns a CatBoost model

frogml.catboost.log_model(
    model=catboost_model,
    repository=repository,
    model_name=name,
    namespace=namespace,
    version=version,
    properties=properties,
    dependencies=dependencies,
    code_dir=code_dir,
)
```

##### Download a CatBoost Model from a Machine Learning Repository

Use the following function to download a deserialized CatBoost model from a Machine Learning repository in Artifactory:

```
import frogml

repository = "repository-name"
name = "model-name"
namespace = "namespace"
version = "version-1"
full_target_path = "full/path/to/target/path"

catboost_deserialized_model = frogml.catboost.load_model(
    repository=repository,
    namespace=namespace,
    model_name=name,
    version=version,
    target_path=full_target_path,
)
```

##### Get Information on a CatBoost Model in a Machine Learning Repository

Use the `frogml.catboost.get_model_info()` function to retrieve information on a specific CatBoost model stored in a Machine Learning repository in Artifactory.

```
frogml.catboost.get_model_info(
    repository=repository,
    name=name,
    namespace=namespace,
    version=version,
)
```

#### 7.21.6.2.2 | Hugging Face Model Type

##### Upload a Hugging Face Model to a Machine Learning Repository

Use the following function to upload a Hugging Face Transformers model to a Machine Learning repository in Artifactory:

```
import frogml

repository = "repository-name"
name = "model-name"
namespace = "namespace"
version = "version-1"
properties = {"key1": "value1"}
```

```
dependencies = ["path/to/dependencies/conda.yaml"]
code_dir = "full/path/to/code/dir"
model = get_huggingface_model() # Function that Returns a huggingface model
tokenizer = get_huggingface_tokenizer() # Function that Returns a huggingface tokenizer

frogml.huggingface.log_model(
    model=model,
    tokenizer=tokenizer,
    repository=repository,
    model_name=name,
    namespace=namespace,
    version=version,
    properties=properties,
    dependencies=dependencies,
    code_dir=code_dir,
)
```

### Download a Hugging Face Model from a Machine Learning Repository

Use the following function to download a deserialized Hugging Face Transformers model from a Machine Learning repository in Artifactory:

```
import frogml

repository = "repository-name"
name = "model-name"
namespace = "namespace"
version = "version-1"
full_target_path = "full/path/to/target/path"

huggingface_deserialized_model = frogml.huggingface.load_model(
    repository=repository,
    namespace=namespace,
    model_name=name,
    version=version,
    target_path=full_target_path,
)
```

### Get Information on a Hugging Face Model in a Machine Learning Repository

Use the following function to retrieve information on a specific Hugging Face Transformers model stored in a Machine Learning repository in Artifactory.

```
import frogml

repository = "repository-name"
name = "model-name"
namespace = "namespace"
version = "version-1"

frogml.huggingface.get_model_info(
    repository=repository,
    name=name,
    namespace=namespace,
    version=version,
)
```

7.21.6.2.3 | ONNX Model Type

### Upload an ONNX Model to a Machine Learning Repository

Use the following function to upload an ONNX model to a Machine Learning repository in Artifactory:

```
import frogml

repository = "repository-name"
name = "model-name"
namespace = "namespace"
version = "version-1"
properties = {"key1": "value1"}
dependencies = ["path/to/dependencies/pyproject.toml", "path/to/dependencies/poetry.lock"]
code_dir = "full/path/to/code/dir"
onnx_model = get_onnx_model() # Function that Returns a onnx model

frogml.onnx.log_model(
    model=onnx_model,
    repository=repository,
    model_name=name,
    namespace=namespace,
    version=version,
    properties=properties,
    dependencies=dependencies,
    code_dir=code_dir,
)
```

### Download an ONNX Model from a Machine Learning Repository

Use the following function to download a deserialized ONNX model from a Machine Learning repository in Artifactory:

```
import frogml

repository = "repository-name"
name = "model-name"
namespace = "namespace"
```

# JFrog Artifactory Documentation Displayed in the header

```
version = "version-1"
full_target_path = "full/path/to/target/path"

onnx_deserialized_model = frogml.onnx.load_model(
    repository=repository,
    namespace=namespace,
    model_name=name,
    version=version,
    target_path=full_target_path,
)
```

## Get Information on an ONNX Model in a Machine Learning Repository

Use the following function to retrieve information on a specific ONNX model stored in a Machine Learning repository in Artifactory.

```
import frogml

repository = "repository-name"
name = "model-name"
namespace = "namespace"
version = "version-1"

frogml.onnx.get_model_info(
    repository=repository,
    name=name,
    namespace=namespace,
    version=version,
)
```

7.21.6.2.4 | scikit-learn Model Type

## Upload a scikit-learn Model to a Machine Learning Repository

Use the following function to upload a scikit-learn model to a Machine Learning repository in Artifactory:

```
import frogml

repository = "repository-name"
name = "model-name"
namespace = "namespace"
version = "version-1"
properties = {"key1": "value1"}
dependencies = ["path/to/dependencies/pyproject.toml", "path/to/dependencies/poetry.lock"]
code_dir = "full/path/to/code/dir"
joblib_model = get_scikit_learn() # Function that Returns a scikit learn model

frogml.scikit_learn.log_model(
    model=joblib_model,
    repository=repository,
    model_name=name,
    namespace=namespace,
    version=version,
    properties=properties,
    dependencies=dependencies,
    code_dir=code_dir,
)
```

## Download a scikit-learn Model from a Machine Learning Repository

Use the following function to download a deserialized scikit-learn model from a Machine Learning repository in Artifactory:

```
import frogml

repository = "repository-name"
name = "model-name"
namespace = "namespace"
version = "version-1"
full_target_path = "full/path/to/target/path"

scikit_learn_deserialized_model = frogml.scikit_learn.load_model(
    repository=repository,
    namespace=namespace,
    model_name=name,
    version=version,
    target_path=full_target_path,
)
```

## Get Information on a scikit-learn Model in a Machine Learning Repository

Use the following function to retrieve information on a specific scikit-learn model stored in a Machine Learning repository in Artifactory.

```
import frogml

repository = "repository-name"
name = "model-name"
namespace = "namespace"
version = "version-1"

frogml.scikit_learn.get_model_info(
    repository=repository,
    name=name,
    namespace=namespace,
```

```
version=version,  
)
```

7.21.6.2.5 | Generic Python Function Model Type

### Upload Generic Python Function Model to a Machine Learning Repository

Use the following function to upload a Generic Python Function model to a Machine Learning repository in Artifactory:

```
import frogml  
  
repository = "repository-name"  
name = "model-name"  
namespace = "namespace"  
version = "version-1"  
properties = {"key1": "value1"}  
dependencies = ["pandas==1.2.3"]  
code_dir = "full/path/to/code/dir"  
  
def simple_regression(x, y):  
    """  
        Perform simple linear regression on the given data.  
  
    :param x: List of input values (independent variable)  
    :param y: List of output values (dependent variable)  
    :return: Tuple containing the slope and intercept of the regression line  
    """  
    n = len(x)  
    mean_x = sum(x) / n  
    mean_y = sum(y) / n  
    # Calculate the slope (m) and intercept (b)  
    numerator = sum((x[i] - mean_x) * (y[i] - mean_y) for i in range(n))  
    denominator = sum((x[i] - mean_x) ** 2 for i in range(n))  
    slope = numerator / denominator  
    intercept = mean_y - slope * mean_x  
  
    return slope, intercept  
  
frogml.python.log_model(  
    function=simple_regression,  
    repository=repository,  
    model_name=name,  
    namespace=namespace,  
    version=version,  
    properties=properties,  
    dependencies=dependencies,  
    code_dir=code_dir,  
)
```

### Download a Generic Python Function Model from a Machine Learning Repository

Use the following function to download a Generic Python Function model from a Machine Learning repository in Artifactory:

```
import frogml  
  
repository = "repository-name"  
name = "model-name"  
namespace = "namespace"  
version = "version-1"  
full_target_path = "full/path/to/target/path" # optional parameter  
  
regression_func = frogml.python.load_model(  
    repository=repository,  
    namespace=namespace,  
    model_name=name,  
    version=version,  
    target_path=full_target_path,  
)
```

### Get Information on a Generic Python Function Model in a Machine Learning Repository

Use the following function to retrieve information about a specific Generic Python Function model stored in a Machine Learning repository in Artifactory.

```
import frogml  
  
repository = "repository-name"  
name = "model-name"  
namespace = "namespace"  
version = "version-1"  
  
frogml.python.get_model_info(  
    repository=repository,  
    name=name,  
    namespace=namespace,  
    version=version,  
)
```

7.21.6.2.6 | PyTorch Model Type

### Upload a PyTorch Model to a Machine Learning Repository

Use the following function to upload a PyTorch model to a Machine Learning repository in Artifactory:

```
import frogml
import torch.nn as nn

repository = "repository-name"
name = "model-name"
namespace = "namespace"
version = "version-1"
properties = {"key1": "value1"}
dependencies = ["pandas==1.2.3"]
code_dir = "full/path/to/code/dir"

class Classifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.hidden1 = nn.Linear(8, 12)
        self.act1 = nn.ReLU()
        self.hidden2 = nn.Linear(12, 8)
        self.act2 = nn.ReLU()
        self.output = nn.Linear(8, 1)
        self.act_output = nn.Sigmoid()

    def forward(self, x):
        x = self.act1(self.hidden1(x))
        x = self.act2(self.hidden2(x))
        x = self.act_output(self.output(x))
        return x

frogml.pytorch.log_model(
    model=Classifier(),
    repository=repository,
    model_name=name,
    namespace=namespace,
    version=version,
    properties=properties,
    dependencies=dependencies,
    code_dir=code_dir,
)
```

### Download a PyTorch Model from a Machine Learning Repository

Use the following function to download a PyTorch model from a Machine Learning repository in Artifactory:

```
import frogml

repository = "repository-name"
name = "model-name"
namespace = "namespace"
version = "version-1"

model = frogml.pytorch.load_model(
    repository=repository,
    namespace=namespace,
    model_name=name,
    version=version,
)
```

### Get Information on a PyTorch Model in a Machine Learning Repository

Use the following function to retrieve information on a specific PyTorch model stored in a Machine Learning repository in Artifactory.

```
import frogml

repository = "repository-name"
name = "model-name"
namespace = "namespace"
version = "version-1"

frogml.pytorch.get_model_info(
    repository=repository,
    name=name,
    namespace=namespace,
    version=version,
)
```

## 7.22 | Maven Repository

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 steps:

1. Resolve Maven Artifacts Through Artifactory
2. Deploy Maven Artifacts

# JFrog Artifactory Documentation

## Displayed in the header

Once your Maven build is configured, Artifactory provides tight integration with commonly used CI servers (such as Jenkins, TeamCity or Bamboo) through a set of plugins that you can install and use freely.

**Read More:**

[JFrog Artifactory and Maven Repositories](#)

[How to set up a Private, Remote and Virtual Maven/Gradle Registry](#)

### 7.22.1 | View Maven Artifacts on Artifactory

Selecting a Maven metadata file (`maven-metadata.xml`) or a POM file (`pom.xml`) from the Tree browsing mode in the Artifact Repository Browser, provides you with details on the selected item.

#### Maven Metadata View

#### POM View

### 7.22.2 | Resolve Maven 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.

#### 7.22.2.1 | Automatically Generate Maven 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.

# JFrog Artifactory Documentation

## Displayed in the header

### Note

The Automatically Generating Settings will allow to you to configure **Virtual Maven repositories only**. Federated repositories cannot be selected. While this is a best practice and the recommended method, you may change any repository by manually editing the `settings.xml` file.

Once you have created your Maven repository, go to **Application | Artifactory | Artifacts**, select your Maven repository, and click **Set Me Up**.

In the Set Me Up dialog, click **Generate Maven Settings**.

You can now specify the repositories you want to configure for Maven.

| Field    | Description                                   |
|----------|-----------------------------------------------|
| Releases | The repository from which to resolve releases |

# JFrog Artifactory Documentation

## Displayed in the header

| Field            | Description                                                                                                                                        |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| 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 <b>Mirror Any</b> Setup |

Once you have configured the settings for Maven, click **Generate Settings** to generate and save the `settings.xml` file.

### 7.22.2.2 | Provision Dynamic Maven 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.

### 7.22.3 | Manually Override the Built-in Maven 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.

### 7.22.4 | Additional Maven Setting: 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.

# JFrog Artifactory Documentation

## Displayed in the header

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>central</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.

### 7.22.5 | Configure Maven 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\}kFposSPUydYZf895y/o4wA==</password>
</server>
<server>
<id>snapshots</id>
<username>admin</username>
<password>\{DESEde\}kFposSPUydYZf895y/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>
<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\}kFposSPUydYZf895y/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`.

# JFrog Artifactory Documentation

## Displayed in the header

### 7.22.5.1 | Force Authentication on Virtual Maven Repositories

Artifactory supports Maven repositories with Allow Anonymous Access enabled by default and will not query the Maven client for authentication parameters.

If you want to enforce authentication you need to explicitly instruct Artifactory to request authentication parameters.

In the New or Edit Repository dialog of the Maven virtual repositories, select the **Force Authentication** check box.

#### Note

When **Force Authentication** is selected, Artifactory will request authentication parameters from the Maven client before trying to access this repository.

### 7.22.5.2 | Force Maven Non-Preemptive Authentication for Local, Remote, and Virtual Repositories

From Artifactory version 7.37.9, an enhanced Maven Authentication mechanism has been implemented in Artifactory to eliminate the need to perform authentication prior to checking if a package is located in local, remote and virtual repositories. With the new authentication mechanism, when reaching Maven-local-three (which requires authentication), instead of first performing for authentication and next authorization. Artifactory will check if the requested item is located in the repository, if the requested package does exist, it will proceed to perform authentication and authorization. If not, a 404 error message will be triggered. This feature is disabled by default.

To enable non-preemptive authentication, add the `artifactory.maven.authentication.nonPreemptive` parameter to the `artifactory.system.properties` file.

#### Reboot Required

You are required to perform a system reboot after adding this flag.

### 7.22.6 | Deploy Maven Artifacts

This section contains the following information:

- Set Up Maven Distribution Management
- Set Up Security in Maven Settings

#### 7.22.6.1 | Set Up Maven 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 **Artifact Repository Browser**, click **Tree** and select the repository you want to deploy to and click **Set Me Up**. The code snippet is displayed under **Deploy**.

**Tip**

Remember that you can not deploy build artifacts to remote, so you should not use them in a deployment element.

7.22.6.2 | Set 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

7.22.7 | Watch the Maven Screencast



7.23 | npm Registry

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:

- The ability to provision npm packages from Artifactory to the npm command line tool from all repository types.
- Calculation of Metadata for npm packages hosted in Artifactory's local repositories.
- Access to remote npm registries (such as <https://registry.npmjs.org>) through Remote Repositories which provide the usual proxy and caching functionality.
- 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.
- Compatibility with the npm command line tool to deploy and remove packages and more.
- Support for flexible npm repository layouts that allow you to organize your npm packages and assign access privileges according to projects or development teams.
- Support for validating remote npm repository metadata.
- SHA512 support for npm packages.

**npm version support**

Artifactory supports npm version 1.4.3 and above. To use npm version 9.0 and above, see npm Login.

7.23.1 | Create npm Repositories

You can create the following npm registries:

# JFrog Artifactory Documentation

## Displayed in the header

- Local npm registry
- Remote npm registry
- Virtual npm registry

### 7.23.1.1 | Set Up Local npm Registry

To enable calculation of npm package metadata in local repositories, in the **Administration** module, go to **Repositories| Repositories | Local** and set the **Package Type** to **npm** when you create the repository.

#### Read More:

[JFrog Artifactory and npm Registries](#)

[How to set up a Private, Remote and Virtual npm Registry](#)

### 7.23.1.1.1 | npm 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 the `npm pack` command. This creates the `.tgz` file for your package which you can then upload to any path within your local npm repository.

### 7.23.1.2 | Set Up Remote npm Registry

A Remote Repositories 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:

1. In the **Administration** module, under **Repositories | Repositories | Remote**, click **New Remote Repository**.
2. In the **New Remote Repository** dialog:
  1. Set the **Package Type** to **npm** and the **Repository Key** value.
  2. Specify the URL to the remote registry in the **URL** field.
3. Click **Save & Finish**.

### 7.23.1.3 | Set Up 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:

1. In the **Administration** module, under **Repositories | Repositories | Virtual**, click **New Virtual Repository**.
2. In the **New Virtual Repository** dialog, set the **Package Type** to **npm**.

# JFrog Artifactory Documentation

## Displayed in the header

3. Select the underlying local and remote npm registries to include in the **Basic** settingstab.

4. Click **Save & Finish** to create the repository.

7.23.1.3.1 | [Advanced npm Virtual Repository Configuration](#)

The fields under **External Dependency Rewrite** are connected to automatically rewriting external dependencies for npm packages that require them.

Field	Description
Enable Dependency Rewrite	When selected, external dependencies are rewritten.
Remote Repository For Cache	The remote repository aggregated by this virtual repository in which the external dependency will be cached.
Patterns Allow List	An Allow 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.  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).

### 7.23.2 | Use 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 Cloud, the URL would be:

- `https://<server name>.jfrog.io/artifactory/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.

### 7.23.2.1 | Use npm Audit

Artifactory now supports `npm audit`, allowing you to get vulnerabilities on your npm projects' dependencies tree.

Audit reports contain information about security vulnerabilities of dependencies and can help fix a vulnerability by providing npm commands and recommendations for further troubleshooting.

This functionality will be enabled by default on npm virtual repositories that aggregate at least one remote repository that supports npm audit. For example, a remote repository that points to <https://registry.npmjs.org> or Artifactory Smart Remote repository.

JFrog Xray users with Artifactory Pro X / Enterprise / Enterprise+ license, will get an enhanced audit report that includes security vulnerabilities from Xray's database. When Xray is configured to work with Artifactory, an audit report can be generated from scratch even without connecting to any remote repository.

Users with *Read Permission* on the npm virtual repository can use the following npm commands:

Command	Description
<code>npm audit</code>	Returns a vulnerability report based on the dependency tree sent by the npm client that is generated by <a href="https://npmjs.com">https://npmjs.com</a> and optionally enhanced by Jfrog Xray.
<code>npm audit fix</code>	Fetches the same report as <code>npm audit</code> and attempts to automatically act upon the recommendations in the report.
<code>npm audit signature</code>	Returns the registry signatures of uploaded packages so you can ensure the integrity of the downloaded packages.

In order to change the source of the npm audit reports, set the `npm.default.audit.provider` system property (default <https://registry.npmjs.org>) to your desired audit provider url.

#### Note

[Learn more about npm audit](#).

### 7.23.2.1.1 | Use npm Audit Signatures

Starting from version 7.83.1, Artifactory supports npm Audit Signatures, a mechanism that applies and verifies artifact signatures using the ECDSA Key Pairs.

When ECDSA auto signing is enabled on a local repository, it creates a signature for each package which allows you to verify its origins using the npm client. You can ensure the integrity of your packages using the following CLI command:

```
npm audit signatures
```

When executing this command, Artifactory will respond with a count of the total number of packages in your repository that are signed.

### 7.23.2.1.1.1 | npm Client Implications for npm Audit Signatures

#### Working With Remote Repositories

When running `npm audit signatures` from a remote repository, the signature process is up to the upstream registry. The official npm registry already signs using ECDSA, therefore the Artifactory provides the signature directly from the remote repository.

Please note that if you are pointing to a different remote repository and want to use the audit signatures command, you must verify that the upstream registry is signing the packages: otherwise, the audit command will always show you exceptions.

#### Working With Virtual Repositories

When running `npm audit signatures` from a virtual repository, make sure to enable [ECDSA signing](#) on all of your local repositories nested under the virtual repository, so that all of the packages will be signed.

In addition, verify that all your remote repositories are pointing to registries that are signing the packages.

#### Avoid mixing signed and unsigned repositories in Virtual repositories

If you mix repositories with signed and unsigned packages, the `npm audit signatures` command will always display an error that you have packages that do not have signatures. This is expected, of course, but will stop the npm client from continuing its usual actions, e.g. proceeding to download packages.

# JFrog Artifactory Documentation Displayed in the header

7.23.2.1.1.2 | [Enable ECDSA Signing in Local Repositories](#)

To enable npm Audit Signatures on a repository:

1. Generate a ECDSA key pair using the following command:

```
gpg --expert --full-generate-key
```

2. Export the public and private keys using the following commands:

```
gpg --output private.pgp --armor --export-secret-key <KEY_ID>
```

```
gpg --output public.pgp --armor --export <KEY_ID>
```

## Note

Replace the placeholder with the key ID that you can find in the output of the key generation command.

3. In the JFrog Platform WebUI, go to **Administration > Platform Security > Keys Management**
4. Click **+ Add Keys**, and select **ECDSA Keys** from the drop-down menu
5. Enter a name and alias for the key you created, and upload the public and private keys. When you are done, click **Add ECDSA Key**
6. Go to the repository page on the JFrog Platform WebUI, and scroll down to the **ECDSA Key Pair** section: under **Primary Key Name**, select the name of the key you just created from the drop down menu.
7. Click **Save** - the repository will automatically reindex and sign all the packages with the keys you provided.

Identify Whether an npm Local Repository Uses ECDSA Signing

To identify whether your local repository has signing enabled, look for the signature in one of the packages in the repository.

You can view the signature for a package under the `dist` object in the package metadata JSON file, for example:

```
"dist":{  
  ..omitted..  
  "signatures": [{  
    "keyid": "SHA256:{SHA256_PUBLIC_KEY}",  
    "sig": "a312b7c3cb4a1b693e8ebac5ee1ca9cc01g2661c14381917dcb111517f72370809..."  
  }],
```

7.23.2.1.1.3 | [Known Issues With npm Audit Signatures](#)

- Running `npm audit signatures` on a virtual repository that includes at least one local or remote repository that does not have signatures enabled would cause the `npm` client to return an error, since not all your packages are signed.
- If you have applied an `include` pattern to the repository with a list of accepted patterns, add the following include pattern to your list to be able to use `npm audit signatures`:  
`.npm/keys.json`

## 7.23.3 | Set the Default npm 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/
```

### Tip

We recommend referencing a [Virtual Repositories 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.

## 7.23.4 | Authenticate the npm Client

Once you have set the default registry, you need to authenticate the `npm` client to Artifactory in one of two ways:

- Running the `npm login` command
- Using basic authentication

### 7.23.4.1 | Authenticate 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 an expirable access token which the client uses for authentication against Artifactory for subsequent and `npm install` `npm publish` actions.

### Note

The npm access token expires within 30 days. To modify the token expiry period, use this Artifactory system property: `artifactory.artifactory.tokens.expiration.timeSecs`, which affects all tokens in Artifactory.

If the token is removed from Artifactory, the client will have to log in again to receive a new token. If the token has expired, run `npm login` again.

### npm-login Limitation

If an Admin creates an access token with a transient user, `npm login` will fail as the user is not registered in Artifactory.

#### 7.23.4.2 | Authenticate npm Using Basic Authentication

To support basic authentication, 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`)

### .npmrc file location

Windows: `%userprofile%/.npmrc`

Linux: `~/ .npmrc`

### Getting .npmrc entries directly from Artifactory

Run the following command to retrieve these strings directly from Artifactory:

```
$ curl -uadmin:<CREDENTIAL> http://<ARTIFACTORY_SERVER_DOMAIN>:8081/artifactory/api/npm/auth
```

Where `<CREDENTIAL>` is your Artifactory password or APIKey.

Here is an example of the response:

```
_auth = YWRtaW46e0RFU2VkJ1u0FRaaXhY0t3bHN4c2RCTViwNjF3PT0=
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 APIKey.

Paste the response to this command in the `~/.npmrc` file on your machine (in Windows, `%USERPROFILE%/.npmrc`).

#### 7.23.5 | Deploy npm Packages (npm Publish)

### Set Your Credentials for npm Publish

The npm command line tool requires that sensitive operations, such as `publish` are authenticated as described under [Authenticating the npm Client](#) above.

### Deploy Your npm 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/
```

#### 7.23.6 | Resolve npm Packages (npm Install)

After adding Artifactory as the default repository you can install a package using the `npm install` command:

### Note

Make sure to replace the placeholder in **bold** with your own package name.

```
npm install <PACKAGE_NAME>
```

To install a package by specifying the Artifactory repository use the following npm command:

### Note

Make sure to replace the placeholders in **bold** with your own package name and repository path.

```
npm install <PACKAGE_NAME> --registry <REPOSITORY_PATH>
```

#### 7.23.6.1 | Resolve npm Packages using dist-tags

You can use `npm dist-tags` to install npm packages from Artifactory.

# JFrog Artifactory Documentation

## Displayed in the header

To install a package using a dist-tag, use the following command:

### Note

Make sure to replace the placeholders in **bold** with your own package name, dist-tag

```
npm install <PACKAGE_NAME>@<DIST-TAG>
```

By default, the `npm install` command will return the first package with the dist-tag that appears in the priority resolution order you have configured. Starting from Artifactory version 7.75.3, you can modify this behavior by using the following system properties, in your Repositories Configurations in Artifactory YAML file:

- To install the latest SemVer version of the package with the dist-tag in your priority resolution repositories, set the `artifactory.npm.merge.latest.dist.tag.base.strategy.enabled=true` system property.
- To install the most recently created version of the package with the dist-tag in your priority resolution repositories, set the following:  
`artifactory.npm.merge.latest.dist.tag.base.strategy.enabled=true` and `artifactory.npm.tag.tagLatestByPublish=true` system properties.

### 7.23.7 | Specify the Latest Version of npm Package

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 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
```

### 7.23.8 | Work with Artifactory npm Registry 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 trace how users interact with your repositories, disable the Allow Anonymous Access setting. This means that users must enter their username and password as described in Setting Your Credentials above.

### 7.23.9 | Use OAuth Credentials in npm

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 the `npm login` command.

### 7.23.10 | Search for npm Packages

Artifactory supports a variety of ways to search artifacts. For details, please refer to [Searching for Artifacts\\_OLD](#).

Artifactory also supports `npm search [search terms ...]`. However, these packages may not be available immediately after being published for the following reasons:

- Local Repositories: When publishing a package to a local repository, Artifactory calculates the search index asynchronously.
- Virtual repositories: 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.

### Tip

Artifactory annotates each deployed or cached npm package with two properties: `npm.name` and `npm.version`.

You can use the [Property Search](#) to search for npm packages according to their name or version.

### 7.23.11 | Clean Up the Local npm Cache

The npm client saves cached 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`.

### 7.23.12 | 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. ApacheTomcat is the HTTP Web Server used by Artifactory, which does not allow encoded slashes by default. This might result in HTTP 400 status codes being returned to the client.

Starting from Artifactory 7.98.9, Artifactory is configured by default to enable encoded slashes, which supports npm scope packages. For more information, see [Artifactory Known Issues](#).

If Artifactory is running behind a reverse proxy, make sure to disable URL decoding on the proxy itself to work with npm scope packages. For Apache, add the following setting inside the `<VirtualHost *:xxx>` block:

AllowEncodedSlashes on directive

### 7.23.13 | Configure npm Client 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 @jfrogs with the registry `http://localhost:8081/artifactory/api/npm/npm-local/` by manually altering your `~/.npmrc` file and adding the following configuration:

```
@jfrogs: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

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"
@jfrogs:registry=http://localhost:8081/artifactory/api/npm/npm-local/
//localhost:8081/artifactory/api/npm/npm-local/:_password=QVA1N050aHZTMnM5Qk02RkR5RjNBVmF4TVF1
//localhost:8081/artifactory/api/npm/npm-local/:username=admin
//localhost:8081/artifactory/api/npm/npm-local/:email=admin@jfrogs.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.

#### Note

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.

### 7.23.14 | Automatically Rewrite External npm 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" Allow List 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:

- Select the **Enable Dependency Rewrite** checkbox in the npm virtual repository advanced configuration.
- Specify an Allow List pattern 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 following example, the external dependencies are cached in the "npm" remote repository and only the package from <https://github.com/jfrogdev> is allowed to be cached.

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
```

#### 7.23.14.1 | Rewrite npm Workflow

1. When downloading an npm package, Artifactory analyzes the list of dependencies required 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 Allow List,
3.
  - 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.
4. 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.

#### SemVer Support

The external dependency rewrite feature for the npm virtual repository supports additional SemVer expressions, such as `semver:4.x.0`.

If you encounter SemVer issues, you can revert the changes using the new feature flag, `artifactory.npm.semver4j.enabled`, by changing its value to false.

#### 7.23.15 | View 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.

#### 7.23.16 | [View npm Build Information](#)

You may store exhaustive build information in Artifactory by running your npm builds with JFrog CLI.

JFrog CLI collects build-info from your build agents and then publishes it to Artifactory. Once published, the build info can be viewed in the Build Browser under [Builds](#).

For more details on npm build integration using JFrog CLI, please refer to [Building npm Packages](#) in the JFrog CLI User Guide.

#### 7.23.17 | [Using Yarn](#)

JFrog Artifactory supports using the Yarn client to easily work against Artifactory npm repositories. Yarn is a package manager that replaces the existing workflow for the npm client and is compatible with the npm registry. It contains the same feature set as the npm while operating faster, more securely, and more reliably.

##### Artifactory support of Yarn

Artifactory 7.x supports and has been tested with Yarn 1.22.4. Previous versions of Artifactory and Yarn have not been tested but should also work sufficiently.

##### 7.23.17.1 | [Set Up Yarn with Artifactory](#)

For your Yarn package manager to work with Artifactory, you first need to authenticate Yarn with Artifactory by updating your npm config file, (i.e the `.npmrc` file), with the credentials from Artifactory as follows:

1. For Yarn 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`):

```
npm config set registry https://artifactory.mycompany.com/artifactory/api/npm/<npm repository name>/
```

2. Once you have set the default registry, you need to authenticate the npm client to Artifactory.

```
npm config set always-auth true
```

3. Run the following command in your npm client. When prompted, provide your Artifactory login credentials:

```
npm login
```

At this point, the `.npmrc` file will be updated and will display the following confirmation.

```
registry=https://artifactory.mycompany.com/artifactory/api/npm/<npm repository name>/
```

```
always-auth=true
```

```
//artifactory.mycompany.com/artifactory/api/npm/<npm repository name>/
```

```
:_authToken=ACCESS_TOKEN
```

You can start using Yarn for installing and publishing packages.

### 7.23.17.2 | Deploy npm Packages Using Yarn

To deploy your package to an Artifactory repository, you can do one of the following:

- Add the following to the package.json file:

```
"publishConfig": {"registry": "https://artifactory.mycompany.com/artifactory/api/npm/<npm repository name>/"}
```

And then run the default yarn publish command:

```
yarn publish
```

- Provide the npm repository you wish to publish to using the yarn publish command as follows:

```
yarn publish --registry https://artifactory.mycompany.com/artifactory/api/npm/<npm repository name>/
```

### 7.23.17.3 | Resolve npm Packages Using Yarn

To install npm packages and add them as dependencies to your package.json, run the following Yarn command:

```
yarn add <PACKAGE_NAME>
```

An example

```
yarn add lodash
```

Alternatively, you can run the following command:

```
yarn add lodash --registry https://artifactory.mycompany.com/artifactory/api/npm/<npm repository name>/
```

### 7.23.17.4 | Work with Scoped Packages in Yarn

You can set up your project to use scoped packages by directing a scope name to your repository. Add the following line to your project's .yarnrc to work with a scoped package. Replace @jfrogs with the relevant scoped package name.

In this example, all Yarn download requests for @jfrogs will be downloaded from Artifactory.

For example:

```
@jfrogs:registry "https://artifactory.mycompany.com/artifactory/api/npm/<npm repository name>/"
```

Resolve npm Scoped Packages

To resolve npm scoped packages, run the following command.

```
yarn add @<scope_name>/package
```

For example:

```
yarn add @jfrogs/project-example
```

### 7.23.18 | Use npm Enforce Path Layout

You can activate a system property to enforce npm path layout. When activated, this system property will prevent, on all your npm repositories, uploading packages with file names that do not match the internal metadata of the package. This ensures that package uploads align with the package name in the artifact path across all your local npm repositories.

Activate this system property as follows:

```
artifactory.npm.enforce.path=true
```

#### Note

Enforcing npm path layout is not backward compatible and is applied only to npm packages that are uploaded after the system property was activated. npm packages that were uploaded to Artifactory prior to activating this system property will continue to be stored in Artifactory even if the file name does not match the internal metadata of the package.

## 7.24 | NuGet Repositories

Artifactory provides complete support for NuGet repositories on top of Artifactory's existing support for advanced artifact management.

Artifactory support for NuGet provides:

- The ability to provision NuGet packages from Artifactory to NuGet clients from all repository types
- Metadata calculation for NuGet packages hosted in Artifactory's local repositories
- The ability to define proxies and caches to access Remote NuGet repositories
- Multiple NuGet repository aggregation through virtual repositories
- APIs to deploy or remove packages that are compatible with NuGet Package Manager Visual Studio extension and the NuGet Command Line Tool
- Debugging NuGet packages directly using Artifactory as Your Symbol Server documentation.
- Support for NuGet API v3 Registries.
- Support for NuGet SemVer 2.0 Package Support.

### Metadata Updates

NuGet Metadata is automatically calculated when adding, moving, copying, or removing NuGet packages.

The calculation is only invoked once the action has been completed. It is also asynchronous and its performance depends on the overall system load, so it may take some time to complete, depending on the number of packages in the repository.

To invoke metadata calculation for a NuGet repository, select **Reindex Packages**.

#### 7.24.1 | Create NuGet Repositories

You can create several types of NuGet repositories:

- Local NuGet Repositories
- Remote NuGet Repositories
- Virtual NuGet Repositories

##### 7.24.1.1 | Set Up Local NuGet Repositories

To create a local repository for which Artifactory will calculate NuGet package metadata, in the **Administration** module, go to **Repositories| Repositories | Local** and select **NuGet** to be the **Package Type**.

###### 7.24.1.1.1 | NuGet 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.

**Read More:** [JFrog Artifactory and NuGet Repositories](#)

#### 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`.

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].[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.

7.24.1.1.2 | Publish to a NuGet 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.

# JFrog Artifactory Documentation

## Displayed in the header

### 7.24.1.2 | Set Up Remote NuGet 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.

#### Note

If you are using NuGet V2, select a Remote registry from the following list of supported registries:

Remote Registry	Download Context Path	Feed Context Path
<a href="https://www.nuget.org">https://www.nuget.org</a>	api/v2/package	api/v2
<a href="https://www.powershellgallery.com">https://www.powershellgallery.com</a>	api/v2/package	api/v2
<a href="https://proget.inedo.com">https://proget.inedo.com</a>	api/v2/package	api/v2
<a href="https://nuget.devexpress.com">https://nuget.devexpress.com</a>	package	" "
<a href="https://community.chocolatey.org">https://community.chocolatey.org</a>	api/v2/package	api/v2
<a href="https://www.nuget.org/packages/Symbol/">https://www.nuget.org/packages/Symbol/</a>	N/A	N/A
<a href="https://msdl.microsoft.com/download/symbols">https://msdl.microsoft.com/download/symbols</a>	N/A	N/A

To create a Remote Repository, in the **Administration** module, go to **Repositories | Repositories | Remote**, and click **New Remote Repository**.

Here are some examples:

- For the NuGet gallery configure the NuGet v3 Feed URL `http://host:port/api/v3/nuget/repoKey` while the URL can stay with `http://host:port/artifactory/api/nuget/repoKey/`. 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`.

#### Note

When accessing the remote <https://www.nuget.org/api/v3/Packages>, you are required to use API v3.

- For NuGet v3 API and metadata configuration it is necessary to configure the NuGet V3 Feed `http://[host:port]/api/v3/nuget/repoKey` while the URL can stay with `http://[host:port]/artifactory/api/nuget/repoKey/`.

# JFrog Artifactory Documentation

## Displayed in the header

- Another Artifactory repository (i.e. a Smart Remote NuGet 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:

### 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 allow list:

1. \*.nuget.org
2. az320820.vo.msecnd.net (NuGet Gallery's current CDN domain)

### 7.24.1.3 | Set Up Virtual NuGet 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 NuGet repository, in the **Administration** module, under **Repositories | Repositories | Virtual**, click **New Virtual Repository** and set **NuGet** to be its **Package Type**. Select the underlying local and remote NuGet repositories to include under the **Repositories** section.

### 7.24.2 | Access 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 Cloud the URL would be:

`https://<server name>.jfrog.io/artifactory/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>`.

# JFrog Artifactory Documentation

## Displayed in the header

This URL handles all NuGet related requests (search, download, upload, delete) and supports both V1 and V2 requests. To use V3 requests, you need to Configure Visual Studio with NuGet v3 API.

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).

### 7.24.3 | Use 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 Cloud the URL would be:

```
https://<server name>.jfrog.io/artifactory/api/nuget/<repository key>
```

#### Note

The NuGet install/delete will not work if there is a subfolder in the command (even if the NuGet push has a subfolder in its command).

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>
```

To use V3 requests, you need to Configure NuGet CLI with NuGet v3 API.

### 7.24.4 | Configure NuGet Authentication

You can configure NuGet authentication using two methods:

- API Key Authentication
- Access Token Authentication

#### 7.24.4.1 | 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
```

### 7.24.4.2 | NuGet Access Token Authentication

You can also authenticate with the server using an Access Token.

1. Create a NuGet repository and follow the setmeup page.
2. Instead of passing the user credentials in the URL, pass the Artifactory Access Token.

Note that the following also use Access Tokens:

```
nuget sources Add -Name <SourceName> -Source <ArtifactoryURL/artifactory/api/nuget/ctie-mobile-snapshots>
```

```
nuget setapikey <ACCESSTOKEN> -Source <SourceName>
```

### 7.24.5 | Anonymous Access to NuGet Repositories

By default, Artifactory allows anonymous access to NuGet repositories. This is defined under **Administration | Security | General Configuration**. For details please refer to Allow Anonymous Access.

#### 7.24.5.1 | Work in NuGet 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>
```

#### Note

NuGet.Config file can also be placed in your project directory, for further information please refer to [NuGet Configuration File](#)

#### 7.24.5.2 | Allow Anonymous Access to NuGet Repositories

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.



When set, Artifactory will first request authentication parameters from the NuGet client before trying to access this repository.

### 7.24.6 | NuGet API v3 Registry Support

#### For .NET and Visual Studio Users: Announcement on Changes to Microsoft's NuGet V3 API.

From February 2021, the [nuget.org](#) team has announced the deprecation of several OData queries resulting in blocking endpoints used by 3rd party clients.

JFrog has evaluated the implication and can conclude that:

- No versions of JFrog Artifactory, that use the NuGet client software, are known to be impacted as this is related to 3rd party clients.
- Artifactory by default uses NuGet V3 endpoints.
- Queries sent by the official NuGet client software will continue to be supported as official NuGet clients do not use those queries and will not suffer from this change.

Artifactory now supports NuGet API v3 feeds and allows you to proxy remote NuGet API v3 repositories (e.g. the [NuGet gallery](#)) and other remote repositories that are set up with the API v3 feed.

To enable API v3, configure the remote repo v3 feed URL value.

#### 7.24.7 | Configure NuGet CLI/ Visual Studio to Work with NuGet v3 API

Manually add the protocolVersion=3 attribute to the NuGet.Config file:

- For Linux installation: The file is located under `~/.config/NuGet/NuGet.Config`
- For a Windows installation: Locate the file usually under `%appdata%\NuGet\NuGet.Config` and add/v3/ to the source URL.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
<packageSources>
    <add key="ArtifactoryV3Remote" value="http://artifactory:8081/artifactory/api/nuget/v3/nuget-remote" protocolVersion="3" />
</packageSources>
...
</configuration>
```

#### 7.24.8 | NuGet SemVer 2.0 Package Support

Artifactory now supports SemVer 2.0 rules for NuGet repositories (for both NuGet API v2 and API v3), which means you can now use pre-release numbers with dot notation or add metadata to the version, for example: `MyApp.3.0.0-build.60`, `MyApp.1.0+git.52406`.

Artifactory serves the requests for downloading packages in SemVer 2.0 rules. For example, if the latest version for a certain package is in SemVer 2.0 convention, Artifactory will return it to the client. NuGet packages with the SemVer 2.0 convention are served from local, remote, and virtual repositories and for both NuGet API v2, and v3 feeds.

NuGet packages with SemVer2 are not available for old NuGet clients (prior to version 4.3.0). This is a breaking change that was made to align with the official global repository behavior. To retain the old behavior, use the `artifactory.nuget.disableSemVer2SearchFilterForLocalRepos` flag.

#### 7.24.9 | View NuGet Build Information

You may store exhaustive build information in Artifactory by running your NuGet builds with JFrog CLI.

JFrog CLI collects build-info from your build agents and then publishes it to Artifactory. Once published, the build info can be viewed under [Builds](#).

For more details on NuGet build integration using JFrog CLI, please refer to [Building NuGet Packages in the JFrog CLI User Guide](#)

#### 7.24.10 | Artifactory as Your Symbol Server

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.

From Artifactory 7.36, you can benefit from the following advanced Symbol Server features:

- Publishing while indexing your Symbol packages to Artifactory from your NuGet Client v3 together with your NuGet packages or as separate Symbol packages
- Resolving Symbol files (.pdb) from virtual and local repositories in the JFrog Platform
- Resolving Symbol files from remote proxies. For example, `http://symbols.nuget.org/download/symbols`.
- Debugging the Symbol files hosted on Artifactory using the Visual Studio debugger tool.

Note that prior to Artifactory 7.36, Symbol Server support was limited to setting Artifactory as a remote Proxy for Symbol files that were hosted as Generic packages in Artifactory

# JFrog Artifactory Documentation

## Displayed in the header

### Support PDB Formats

- Microsoft PDB V7 (Microsoft C/C++ MSF 7.00)
- Portable PDB v1.0

#### 7.24.10.1 | Publish NuGet Symbol Packages to Artifactory

The JFrog Platform supports publishing and automatically indexing your Symbol packages to be consumed by the debugger.

Prerequisite:

- Create your NuGet Symbol packages (.snupkg). For more information, see [Create Symbol Packages](#).

#### 7.24.10.1.1 | Set up a Local Symbol Server Repository

Local repositories enable you to deploy NuGet Symbol (.snupkg) packages. Artifactory calculates the metadata for all the Symbol packages and indexes them to allow users to download Symbol files through the Visual Studio debugger.

To create a NuGet Symbol local repository:

1. Navigate to the **Administration** module, go to **Repositories | Repositories | Local | New Local Repository**
2. Select **NuGet** as the **Package Type**.

#### 7.24.10.1.2 | Set up a Remote Symbol Server Repository

You can proxy a remote Symbol Server through the JFrog Platform remote repositories. A Remote Repository defined in Artifactory serves as a caching proxy for a registry managed at a remote URL such as (which is the `http://symbols.nuget.org/download/symbols`).

Symbol files requested from a remote repository are cached on demand. You can remove downloaded Symbol files from the remote repository cache; however, you can not manually push Symbol packages to the remote NuGet repository.

To define a remote repository to proxy as a remote Symbol Server follow these steps:

1. From the **Administration** module, select **Repositories | Repositories | Remote**.
2. Click **New Remote Repository** and select **NuGet** from the Select Package Type dialog.
3. In the Basic tab, set the **Repository Key** value, and specify the URL to the remote registry in the **NuGet Symbol Server URL** field. Note that the default is set to `http://symbols.nuget.org/download/symbols`.

#### 7.24.10.1.3 | Set up a Virtual Symbol Server Repository

An Artifactory Virtual Repository aggregates packages from both local and remote repositories.

This allows you to access both locally-hosted NuGet Symbol packages and remote-proxied NuGet Symbol files from a single URL that is defined for the virtual repository.

To create a virtual NuGet repository:

1. In the **Administration** module, under **Repositories | Repositories | Virtual**, Click **New Virtual Repository** and set **NuGet** to be its **Package Type**.
2. Select the underlying local and remote NuGet Symbol repositories to include under the **Repositories** section.

# JFrog Artifactory Documentation

## Displayed in the header

### 7.24.10.2 | Configure the NuGet CLI to Work Opposite Artifactory as the Symbol Server

In addition to building and creating NuGet packages (.nupkg), the NuGet client also supports creating associated Symbol packages (.snupkg or .symbols.nupkg), that contain all the relevant symbol files for the NuGet package. The Symbol packages can be pushed to a Symbol Server, where the Symbol files can be indexed and consumed by the Visual Studio Debugger.

The Symbol package structure is similar to the NuGet package but contains the Symbol files instead of the source files.

To configure the NuGet CLI:

1. In the JFrog Platform, navigate to **Application Module | Artifactory | Artifacts**.
2. Select the NuGet repository you created,
3. Select **Set Me Up**.
4. In the **Configure** tab, set up the NuGet repository to work against the NuGet Client.

- 
5. Add the following line to theNuGet.configfile.

6. In the **Deploy** tab, choose from one of the following Push options.

- Push NuGet packages together with their related Symbol packages

When you run the `nuget push` command, if there is a Symbol package present in the same directory, then the Symbol package will be automatically pushed to the same location.

As displayed in the following example.

```
nuget push mypackage.1.0.0.nupkg -Source ArtifactoryNuGetV3
```

# JFrog Artifactory Documentation

## Displayed in the header

- **Pushing NuGet Symbol packages only** When you run the nuget push command, you can decide to push only Symbol packages by adding an 's' as the prefix to the nupkg string.

As displayed in the following example.

```
nuget push mypackage.1.0.0.snupkg -Source ArtifactoryNuGetV3
```

### 7.24.10.3 | View Individual Symbol Package Information

After deploying your Symbol package to Artifactory, you can view the indexed Symbol files. In the [Artifact Browsing](#), select your **NuGet repository** and scroll down to find and select the Symbol files you want to inspect.

### 7.24.10.4 | Debug Symbol Files in Visual Studio

The way in which Visual Studio and other debugging tools match an assembly and PDB file, is by using the *assembly hash*. This hash is stored in the .dll and .pdb files and must match for the debugging and source stepping to work.

When a .nupkg contains .pdb files, Visual Studio will **not** reach out to MyGet to download Symbols and sources. When trying to debug using this type of package, Visual Studio will find the .pdb on the disk instead of reaching out to MyGet to download it, and therefore will fail to step into the code because of that.

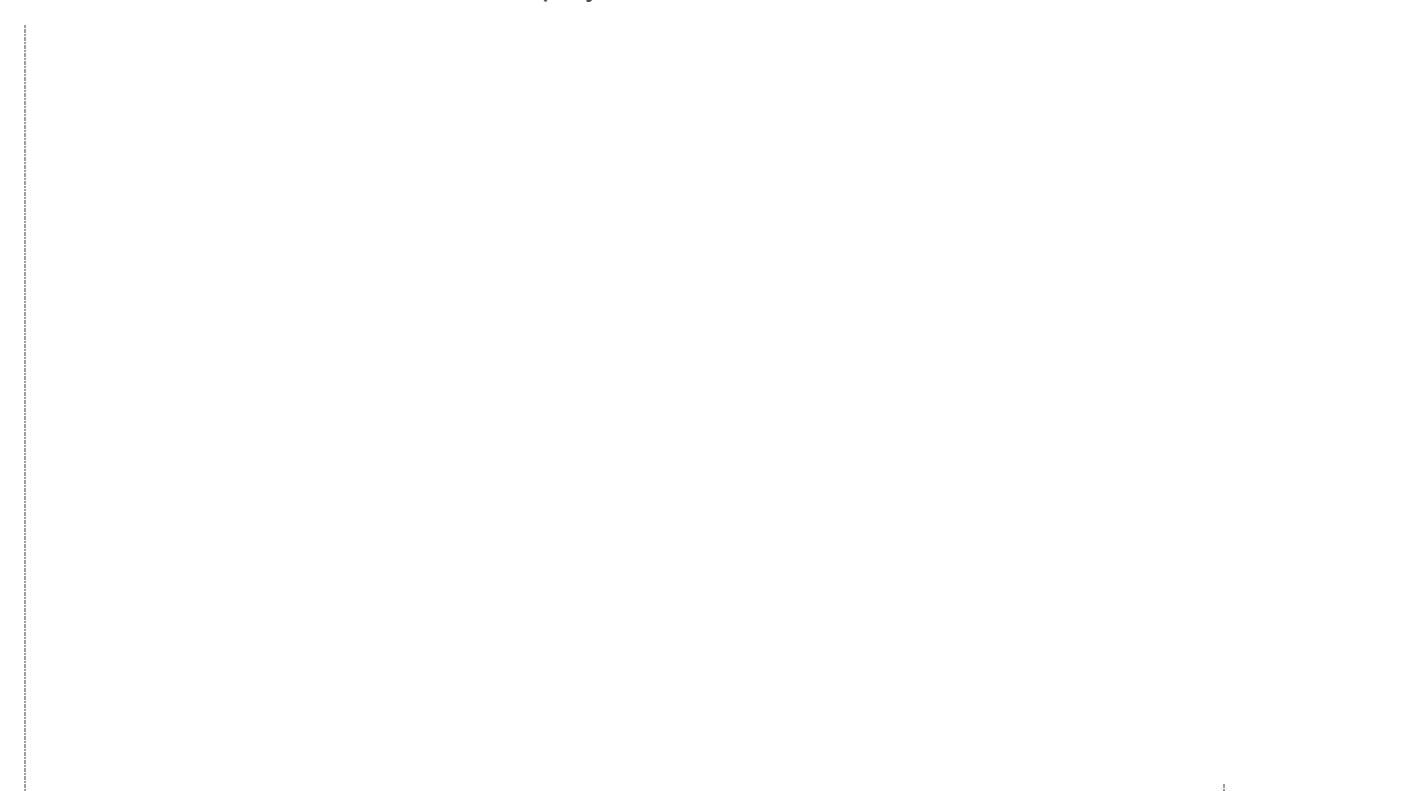
#### Prerequisites

1. In Visual Studio, under **Tools | Options (or Debug | Options) | Debugging | General**, clear the **Enable Just My Code** field.
2. Set Artifactory to be your Symbol Server in Visual Studio by going to **Tools | Options | Debugging | Symbols** and adding the virtual or local repository URL path.

To debug the Symbol files in Artifactory:

1. Run the Visual Studio Debugger and type in your credentials.

Once logged in, the debugger scans the local cache and then goes to the virtual repository in Artifactory as displayed in the following example.



2. Once the symbol is resolved by Visual Studio, proceed to debug the deployed Symbol file.

7.24.11 | [Watch the NuGet Screencast](#)



7.25 | [NVIDIA NIM Repositories](#)

- To start working with NVIDIA NIM, see [Get Started with NVIDIA NIM](#)
- To configure your NIM Model client to work through Artifactory, see [Set Up NIM Model client](#)

The JFrog Artifactory integration with NVIDIA NIM allows you to cache NVIDIA NIM models in Artifactory via the remote repository.

NVIDIA NIM is a set of easy-to-use microservices for accelerating the deployment of foundation models on any cloud or data center and helps keep your data secure. NIM has production-grade runtimes, including ongoing security updates. Run your business applications with stable APIs backed by enterprise-grade support.

# JFrog Artifactory Documentation

## Displayed in the header

NGC Catalog is the public registry for NVIDIA NIM Models.

To learn more about NVIDIA NIM Models, refer to Deploy Generative AI With NVIDIA NIM.

### NVIDIA NIM Repositories

**NVIDIA NIM Remote Repositories:** Remote repositories in Artifactory act as proxies for repositories on remote servers. Artifactory first checks its local cache for the requested package. If the package is not found in the cache, Artifactory retrieves it from the remote repository via the Internet. Once retrieved, the file is cached locally in Artifactory, making it available for future requests without an internet connection. Importantly, only the requested package is cached, not the entire remote repository.

Artifactory supports proxying remote NVIDIA NIM registries through remote repositories. The Remote Repository in Artifactory is a caching proxy for a registry managed at a remote URL <https://api.ngc.nvidia.com/>.

Resources that are requested from a remote repository are cached on demand. You can remove downloaded resources from the remote repository cache. However, you cannot manually push resources to a remote repository.

### Main Features of NVIDIA NIM in Artifactory

The following are the main features of NVIDIA NIM in Artifactory:

- **Single Source of Truth:** This integration allows users to work with NVIDIA NIM consumption and Enterprises' best practices for external artifact consumption via a single source of truth in Artifactory.
- **Consistent and Reliable Access to Remote NVIDIA NIM Models:** Remote NVIDIA NIM Repositories in Artifactory proxy external resources from the NGC Catalog and cache downloaded NVIDIA NIM Models. This reduces overall networking and creates fast, consistent, reliable access to NIM Models on these remote resources.

### NVIDIA NIM Supported Clients

Docker

### Note

Currently, we do not support **NGC CLI** and **NGC SDK**.

### Limitations of NVIDIA NIM in Artifactory

Currently, JFrog Artifactory does not support the following with NVIDIA NIM packages:

- Xray
- Curation
- Local and Virtual Repositories
- Anonymous Access
- Direct Cloud Storage Download

### Note

Supports NIMs with Version 1.3 or later

### 7.25.1 | Get Started with NVIDIA NIM

This topic outlines getting started with NVIDIA NIM repositories in Artifactory.

1. Create NVIDIA NIM Remote Repository
2. Configure NIM Client
3. Resolve NVIDIA NIM Models

Resolving NVIDIA NIM Models with Artifactory

7.25.2 | Create NVIDIA NIM Remote Repository

This section describes the steps and references to create NVIDIA NIM remote repository and configure the repositories.

NIM Repository Prerequisites

- NVIDIA Personal Key

- Remote Docker repository pointing to the URL <https://nvcr.io>.

NVIDIA NIM models are provided as prebuilt containers packaged with optimized models.

To resolve a model from NVIDIA NIM to a remote **Docker** repository, ensure that the repository is configured to download NVIDIA NIM containers. The URL parameter should be set to <https://nvcr.io/>, as NIMs are packaged as container images for individual models or model families. This URL points to the remote **Docker** repository from which the container can be downloaded.

The authentication for the remote Docker repository:

- **Username:** \$oauthToken
- **Password / Access Token:** NVIDIA Personal Key

To create a remote repository to proxy a remote NVIDIA NIM registry, follow these steps:

1. From the Administration module, click **Repositories**.
2. Click **Remote Repository** from **Add Repositories**.

3. Click **NVIDIA NIM Models** in the **Select Package Type** dialog.

4. Set the **Repository Key**.

**Note**

- **URL** is auto-populated: <https://api.ngc.nvidia.com>
- **Username**: Leave this field blank
- **Password / Access Token**: NVIDIA Personal Key, generated as a prerequisite

5. For other Basic settings, refer to the [Basic Settings for Remote Repositories](#).

6. For Advanced settings, refer to the [Advanced Settings for Remote Repositories](#).

7. Click [Create Remote Repository](#).

### 7.25.3 | Set Up NIM Model client

This topic describes how to view NIM repository set-me-up instructions. It provides instructions for the created NIM repositories to view helpful code snippets from the UI to resolve NIM models using the docker client.

To view NVIDIA Set Me Up instructions based on your repository selection, follow these steps:

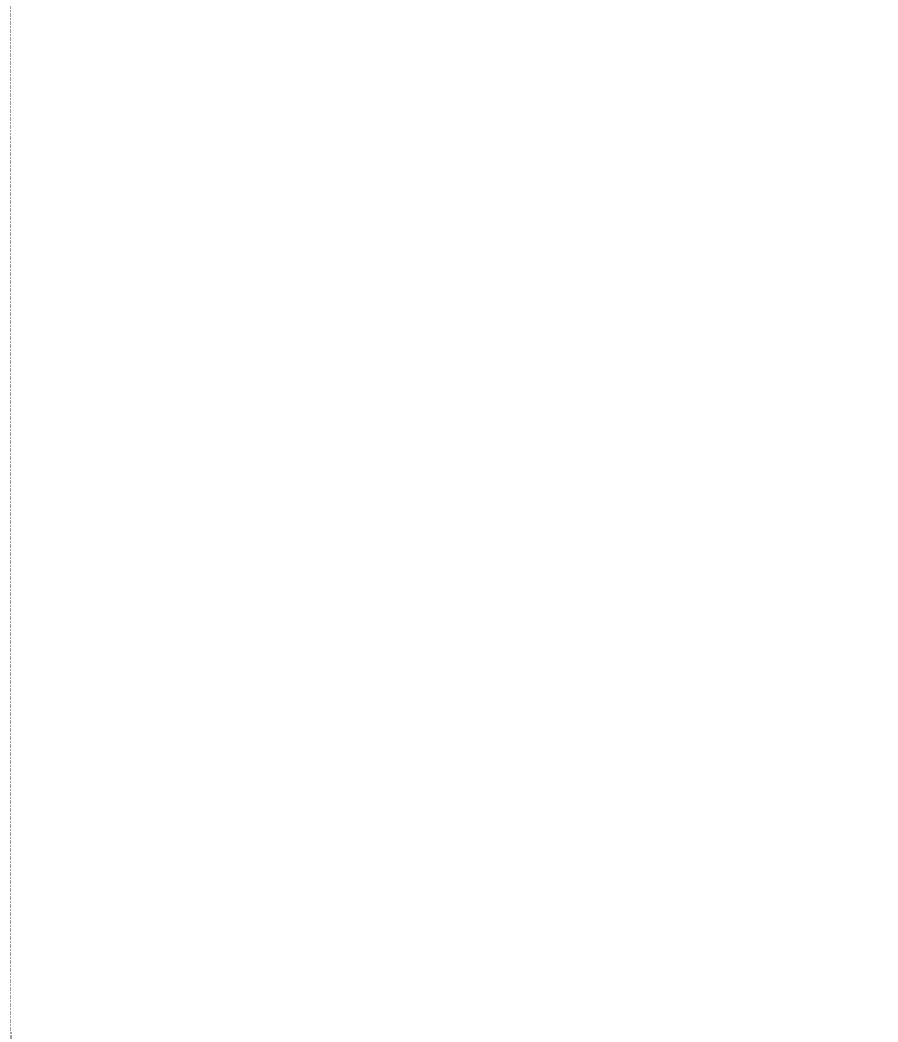
1. Click **Application** tab, and then click **Artifacts** under **Artifactory**.
2. Locate the repository you want to configure, and then click **Set Me Up**.

**Note**

Alternatively, you can choose the repository from the **Repository** drop-down list under **Set Up A NIM Model Client** modal.

3. Enter your JFrog account password to generate a token, and then click **Generate Token & Create Instructions**.
4. Click one of the following as applicable:

- Configure tab



- Resolve tab



#### 7.25.3.1 | Configure NIM Client

This topic describes configuring the NIM Model client to point to remote repositories in Artifactory. It provides instructions for configuring the client to use the NIM remote repository that points to <https://api.ngc.nvidia.com>.

To set up the NIM client to work with Artifactory, follow these steps:

##### 1. Set Environment Variables

Open your terminal and set the following environment variables to configure access to the NVIDIA NIM repository:

##### Note

Note: Make sure to replace the placeholders in bold with the appropriate values as follows:

The following fields are populated by the user interface (UI):

# JFrog Artifactory Documentation

## Displayed in the header

- <TOKEN>: Artifactory Repository token
- <DOMAIN>: Use the domain without https://
- <REPO>: The nim remote repository key
- <PROTOCOL>: http | https
- ./: Specify the directory on your local machine

```
export NGC_API_KEY=<TOKEN>
export NGC_API_ENDPOINT=<DOMAIN>/artifactory/api/nimmodel/<REPO>
export NGC_AUTH_ENDPOINT=<DOMAIN>/artifactory/api/nimmodel/<REPO>
export NGC_API_SCHEME=<PROTOCOL>
export CACHE_DIR=./
```

- **NGC\_API\_KEY=<TOKEN>**: Your Artifactory repository token for authentication, generated by the **Set Me Up**. Replace <TOKEN> with your actual token.
- **NGC\_API\_ENDPOINT=<DOMAIN>/artifactory/api/nimmodel/<REPO>**: The URL of the remote NIM repository. Replace <DOMAIN> with the repository domain and <REPO> with the model repository name.
- **NGC\_API\_AUTH\_ENDPOINT=<DOMAIN>/artifactory/api/nimmodel/<REPO>**: The authentication URL for the NIM repository. Replace <DOMAIN> with your repository's domain and <REPO> with the model repository name.
- **NGC\_API\_SCHEME=<PROTOCOL>**: The protocol to use for communication. Set to **http | https** for secure connections.
- **CACHE\_DIR=./**: The local directory for storing cached files (for example, downloaded models). Replace ./ with the desired path.

### 2. Authenticate with Docker Repository

Log in to your Artifactory Docker repository using Docker. In your terminal, enter:

```
docker login <DOMAIN>
```

You will be prompted to enter your Artifactory username and password or API key.

**For example:**

```
docker login awesome.jfrog.io
```

#### Note

For alternative Docker authentication methods, such as manually configuring your credentials or using different clients or versions, refer to the **Set Me Up** instructions for the Docker repository.

### 7.25.3.2 | Resolve NVIDIA NIM Models

This topic describes how to resolve NIM Models using NIM remote repositories in Artifactory. It provides instructions to resolve packages using NIM remote repository that points to <https://api.ngc.nvidia.com>.

To run the NIM model Docker container, execute the following command in your terminal.

The sample snippet structure is as follows:

#### Note

Make sure to replace the placeholders in bold with appropriate values.

The following fields require your input:

- **[docker-repository]**: The NIM docker remote repository key. The Docker image for the model will be pulled from the Docker repository.
- **[image-path]**: The path of the NIM image
- **[tag]**: The tag of the NIM image

```
docker run -it --rm --gpus all \
-e NGC_API_KEY="$NGC_API_KEY" \
-e NGC_API_ENDPOINT="$NGC_API_ENDPOINT" \
-e NGC_API_SCHEME="$NGC_API_SCHEME" \
-e NGC_AUTH_ENDPOINT="$NGC_AUTH_ENDPOINT" \
-v "$CACHE_DIR:/opt/nim/.cache" \
[<docker-repository>]/[<image-path>]:[<tag>]
```

This docker run command runs a Docker container with various configurations and parameters. Here's a breakdown of each parameter:

# JFrog Artifactory Documentation

## Displayed in the header

- **docker run:** This is the base command to run a Docker container. It starts the container, and you can specify various options to configure the environment and behavior.
  - **-it:**
    - **-i** stands for interactive, which keeps the standard input (stdin) open, allowing you to interact with the container if needed.
    - **-t** allocates a pseudo-TTY, which provides terminal features like color and text formatting inside the container.
- Together, **-it** allows for interactive use of the container in the terminal (for example, if you want to run commands inside it).
- **--rm:** This option automatically removes the container once it exits. It is useful for keeping your system clean by ensuring that stopped containers don't remain on your system.
  - **--gpus all:** This option allows Docker to utilize all available GPUs on your machine. The container will have access to all the GPUs on the host system (assuming the Docker daemon supports GPU usage, typically with NVIDIA Docker or CUDA support).
  - **-e NGC\_API\_KEY="\$NGC\_API\_KEY":** This sets an environment variable NGC\_API\_KEY inside the container. The value of NGC\_API\_KEY is taken from the host system's environment variable \$NGC\_API\_KEY. This is commonly used to provide API keys or secrets.
  - **-e NGC\_API\_ENDPOINT="\$NGC\_API\_ENDPOINT":** This sets an environment variable, NGC\_API\_ENDPOINT, inside the container. The value is taken from the host system's \$NGC\_API\_ENDPOINT. It might be used to set the endpoint for some API interactions (for example, a cloud service or registry).
  - **-e NGC\_API\_SCHEME="\$NGC\_API\_SCHEME":** This sets an environment variable NGC\_API\_SCHEME inside the container. The value is taken from the host system's \$NGC\_API\_SCHEME.
  - **-e NGC\_AUTH\_ENDPOINT="\$NGC\_AUTH\_ENDPOINT":** This sets the NGC\_AUTH\_ENDPOINT environment variable to the value of NGC\_AUTH\_ENDPOINT.
  - **-v "\$CACHE\_DIR:/opt/nim/.cache":** This binds a volume (shared directory) between the host and the container. The host's directory "\$CACHE\_DIR" is mapped to /opt/nim/.cache inside the container. This allows data or files to be shared between the host and container, or to persist data outside the container after it exits.
    - \$CACHE\_DIR is an environment variable from the host that represents the path to a cache directory.
- [**<docker-repository>**][**<image-path>**]:[**<tag>**]: This specifies the Docker image to run. It consists of three parts:
    - **<docker-repository>**: The specified Docker repository from which the model will be pulled.
    - **<image-path>**: The specific path to the image within the repository. For example, nim/meta/llama-3.1-8b-instruct.
    - **<tag>**: The version or tag of the Docker image. This could be a version number, such as the latest, 1.1.0, or a specific commit ID.

To get the container image and tag, follow these steps:

1. From the **NGC Catalog**, click **Container**.
2. Search for your desired container, locate it, and then click on the container.



3. Click **Get Container** and copy the image path (the image path refers to the part of the string that starts after nvcr.io/ and ends just before the :)

For example, nim/meta/llama-3.1-8b-instruct

4. To view all tags, click **View all tags**.



**For example:** If you wanted to run a llama Docker image with GPU support, it could look like

```
docker run -it --rm --gpus all \
-e NGC_API_KEY="$NGC_API_KEY" \
-e NGC_API_ENDPOINT="$NGC_API_ENDPOINT" \
-e NGC_API_SCHEME="$NGC_API_SCHEME" \
-e NGC_AUTH_ENDPOINT="$NGC_AUTH_ENDPOINT" \
-v "$CACHE_DIR:/opt/nim/.cache" \
-nim-docker-remote/nim/meta/llama-3.1-8b-instruct:1.1.0
```

It runs the llama container with GPU support, using environment variables for configuration, mounting a cache directory, and ensuring proper permissions.

### 7.25.4 | Advanced Topics - NIM Repository

This section outlines all the other topics related to NIM repositories in Artifactory. It includes NIM repository layout and managing NIM repositories and models.

The following are the related topics:

- NIM Repository Layout
- Manage NVIDIA NIM Remote Repositories
- Manage NVIDIA NIM Models

#### 7.25.4.1 | NIM Repository Layout

This topic describes the NIM remote repositories layout.

##### NIM Docker Repository Layout

The following is the NIM Repository Layout.

```
<repository-name>
  └── .jfrog
      ├── caller-info.json
      └── model-mapping.json
  └── models
      └── org
          └── nim
              └── team
                  └── meta
                      └── <model_name>
                          └── <model_version>
                              └── .jfrog
                                  └── checksums.blake3
                              └── config.json
                              └── generation_config.json
```

- **<repository-name>**: The root folder of your Docker remote repository.
- **.jfrog**: Contains JFrog internal configuration files for integration.
- **caller-info.json**: JSON metadata typically includes information about API callers or services interacting with the repository.
- **model-mapping.json**: A file that maps various models to their paths or specifications.
- **models**: The main directory for machine learning models.
- **org**: A directory that encapsulates organizational structure.
- **nim**: Indicates a specific technology or model framework
- **team**: Represents a specific team working within the organizational folder.
- **meta**: Contains metadata and model records.
- **<model\_name>**: Each model is stored in its directory.
- **<model\_version>**: Different versions of the model (versioning is crucial for deployment).
- **.jfrog**: Contains files specific to model configuration.
- **checksums.blake3**: A file with checksum values for integrity verification of the model files.
- **config.json**: Contains model-specific configurations
- **generation\_config.json**: Configuration for generation-specific settings

##### NIM Remote Repository Layout

```
<repository-name>
  └── .jfrog
      ├── caller-info.json
      └── model-mapping.json
  └── models
      └── org_name
          └── team_name
              └── <model_name>
                  └── <model_version>
                      └── .jfrog_nim_model_info.json
                      └── <model>
                          └── <FILE_1>
                          └── <FILE_2>
```

# JFrog Artifactory Documentation

## Displayed in the header

- <repository name>: The top-level name of the repository where all models are stored. This often reflects the purpose or organization managing the models.
- <models>: A directory that contains all the models.
- <org\_name>\_<team\_name> - A folder naming convention where:
  - <org\_name>: represents the organization's name (for example, my\_org).
  - <team\_name>: represents the specific team within the organization (for example, data\_science).Together, they are separated by an underscore (for example, my\_org\_data\_science).
- <model\_name> - The name of the specific model (for example, image\_classifier).
- <model\_version> - Version of the model, usually following semantic versioning (for example, v1.0.0).
- \_jfrog\_nim\_model\_info.json - A metadata file in JSON format that contains information about the model, such as its name, version, description, authorship, and any dependencies or requirements for the model.
- <model> - A directory that contains the actual model files necessary for deployment or inference.
- <FILE\_1>, <FILE\_2> - These are placeholders for specific model files, which could include:
  - The serialized model itself (for example, a .pt file for PyTorch, .h5 for Keras).
  - Configuration files.
  - Any other relevant resources

### 7.25.4.2 | Manage NVIDIA NIM Remote Repositories

This section outlines how to edit repository configurations and delete NIM remote repositories in Artifactory. You can edit existing repositories to properly configure them to meet your needs and delete unintended repositories.

The following are the related topics:

- [Edit NVIDIA NIM Remote Repository](#)
- [Delete NVIDIA NIM Remote Repository](#)

#### 7.25.4.2.1 | Edit NVIDIA NIM Remote Repository

This topic describes how to edit NIM repositories in Artifactory. You can edit repository settings for NIM remote repositories, ensuring they are properly configured to meet your needs.

**To edit NVIDIA NIM remote repository, follow these steps:**

1. From the Administration module, click **Repositories**.
2. Click **Remote** in the **Repositories** window.
3. Click the NVIDIA NIM Repository you want to edit from the list.
4. Edit the fields you want to make changes.
5. Click **Save**.

#### 7.25.4.2.2 | Delete NVIDIA NIM Remote Repository

This topic describes how to delete NIM remote repositories in Artifactory. You can delete unintended NIM repositories.

**To delete NVIDIA NIM remote repository, follow these steps:**

1. From the Administration module, click **Repositories**.
2. Click **Remote** in the **Repositories** window.
3. Hover the mouse pointer over the repository you want to delete, click ... icon, and then click **Delete** icon.

## 7.25.5 | Manage NVIDIA NIM Models

This section outlines how to manage NIM Models. You can search and list NIM models in Artifactory.

The following are the related topics:

- [Search NVIDIA NIM Models](#)
- [List NVIDIA NIM Model versions/tags](#)

### 7.25.5.1 | Search NVIDIA NIM Models

This topic describes how to search NIM Models in Artifactory. It provides instructions to search models using a number of parameters.

You can search for NVIDIA NIM Models by name using the [Artifact Package Search](#) or through the REST API.

#### via UI

Artifactory supports a variety of ways to search for artifacts. For details, please refer to [Browse and Search Artifacts](#). A new NVIDIA NIM Model will only be found once Artifactory checks for it according to the Retrieval Cache Period setting.

#### via API

Search Artifactory using the REST API. Refer to [Artifact Search API](#).

#### Tip

Artifactory annotates each cached NVIDIA NIM model with the following properties:

`Nimmodel.createdDate, Nimmodel.displayName, Nimmodel.framework, Nimmodel.labels, Nimmodel.modelFormat, Nimmodel.latestVersionIdStr, Nimmodel.name, Nimmodel.precision, Nimmodel.publisher, Nimmodel.shortDescription, Nimmodel.updatedDate and Nimmodel.versionId`

Property Search can search for NVIDIA NIM Models according to their name or version.

### 7.25.5.2 | List NVIDIA NIM Model versions/tags

This topic describes how to list NIM model versions in Artifactory. It provides references to work with listing package versions.

To learn more about list versions, refer to the [Viewing Package Information](#).

## 7.26 | OCI Registry

Starting from Artifactory version 7.75.3, The JFrog Artifactory integration with Open Container Initiative (OCI) allows you to manage OCI artifacts in Artifactory. OCI is an open-source container governance structure that sets clear industry standards for containers. These standards allow different containerized applications to work together seamlessly.

### 7.26.1 | Main Features of OCI Registry in Artifactory

#### Single Point of Truth

This integration allows users to work with OCI registry, manage and save your OCI containers in Artifactory while providing full flexibility and usability.

#### Use OCI Natively

Artifactory supports the relevant calls of the OCI Distribution specification API so that you can transparently use the OCI client to access images through Artifactory.

#### Secure Private OCI Registry with Fine-Grained Access Control

Local OCI repositories are where you store internal OCI images for distribution across your organization. With the fine-grained access control provided by built-in security features, Artifactory offers secure OCI push and pull with local OCI repositories as fully functional, secure, private OCI registries.

#### Consistent and Reliable Access to Remote Images

Remote OCI Repositories in Artifactory proxy external resources such as DockerHub or a remote OCI repository in another Artifactory instance, and cache downloaded images. This reduces overall networking and creates fast, consistent, and reliable access to images on these remote resources.

#### Supported Media Types

The OCI integration with Artifactory supports the following media types:

`"application/vnd.oci.image.manifest.v1+json"`  
`"application/vnd.oci.image.index.v1+json"`

“application/vnd.docker.distribution.manifest.v2+json”  
“application/vnd.docker.distribution.manifest.list.v2+json”

### 7.26.2 | Set Up an OCI Repository

#### Supported Naming Conventions

Due to a limitation in the OCI client, you cannot use underscores (\_) when naming OCI repositories. For example, the OCI client will not be able to communicate with a repository named test OCI\_repo, but it will work with a repository named test.OCI.repo.

You can create the following OCI repository types:

- Local repositories
- Remote repositories
- Virtual repositories

#### 7.26.2.1 | Set Up Local OCI Repositories

A local OCI repository is where you deploy and host your internal OCI resources. It is, in effect, an OCI registry able to host collections of tagged OCI resources which are your OCI 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 create a local OCI repository:

1. From the **Administration** module, select **Repositories | Repositories | Local**.
2. Click **New Local Repository** and select **OCI** from the **Select Package Type** dialog.
3. Set the **Repository Key**.
4. (Optional) Set **Max Unique Tags**. This specifies the maximum number of unique tags, per repository, that should be stored for a OCI 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.
5. (Optional) Set **Tag Retention**. This specifies the number of tags that the JFrog platform will retain when they are overwritten. Leaving the field at 1 (default) means that no overwritten tags will be saved.

#### 7.26.2.2 | Set Up Remote OCI Repositories

Artifactory supports proxying remote OCI registries through **remote repositories**. A Remote Repository in Artifactory serves as a caching proxy for a registry managed at a remote URL such as <https://registry-1.docker.io/>, or even an OCI repository managed at a remote site by another instance of Artifactory. Since there is no default registry for OCI, DockerHub is used as the default as a popular registry.

Resources that are requested from a remote repository are cached on demand. You can remove downloaded resources from the remote repository cache, however, you can not manually push resources to a remote repository.

#### Preventing DockerHub Remote Repository Restrictions

Unauthenticated users on DockerHub are blocked once reaching the download rate limit of 100 pulls per six hours. To prevent reaching this limit, please authenticate using your DockerHub credentials.

To create a remote repository to proxy a remote OCI registry, follow these steps:

1. From the **Administration** module, select **Repositories | Repositories | Remote**.
2. Click **New Remote Repository** and select **OCI** from the **Select Package Type** dialog.
3. In the Basic tab, set the **Repository Key** value, and specify the URL to the remote registry in the **URL** field.  
  
If you are proxying DockerHub, use <https://registry-1.docker.io/> as the URL, and make sure the **Enable Token Authentication** checkbox is selected (these are the default settings).

Alternatively, to use your Docker account type, you need to authenticate the DockerHub pull requests, by setting your user and password in your basic OCI repository.

4. To allow Artifactory to download foreign layers to the OCI remote repository, click the **Advanced** tab and select the **Enable Foreign Layers Caching** checkbox.
5. (Optional) To match external URL when downloading from foreign layers, set include/exclude patterns. These are Ant-style expressions that allow you to specify where foreign layers may be downloaded from: supported expressions include \*, \*\*, ?.

For example, setting the pattern `**/github.com/**` will allow the downloading of foreign layers only from the github.com host.

The default setting for this field is `**`, which allows the download of foreign layers from any source.

6. Configure the network settings. For more information, see [Network Settings](#).

7. Click **Save and Finish**.

#### Smart Remote Repository Path and Domain

When accessing a Smart Remote OCI repository through Artifactory, the repository URL must be prefixed with `api/oci` in the path:

#### Tip

Make sure to replace the placeholder with your JFrog domain information.

`<YOUR_JFROG_DOMAIN>/artifactory/api/oci/<REPOSITORY>/v2`

For example:

# JFrog Artifactory Documentation Displayed in the header

<https://my-awesome-jfrog.io:8081/artifactory/api/oci/jfrog-oci-remote/v2>

## 7.26.2.3 | Set Up Virtual OCI Repositories

Artifactory supports virtual OCI Repositories. A **Virtual Repository** 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 OCI repositories, as well as remote images that are proxied by remote OCI 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, and replace the default deployment target, while the changes will be transparent to the users.

To create a virtual OCI repository:

1. From the Administration module, select **Repositories | Repositories | Virtual**.
2. Click **New Virtual Repository** and select **OCI** from the **Select Package Type** dialog.
3. Set the **Repository Key** value.
4. Select the underlying local and remote OCI repositories to include under the **Repositories** section.
5. (Optional) Configure your **Default Deployment Repository**, where the images you upload to this virtual repository will be routed. Once you configure this repository, your OCI repository will be a fully-fledged OCI repository.

Additionally, you can set up your virtual repository to wrap a series of repositories representing the stages of your development pipeline, and then use them to promote resources from the default deployment repository through the pipeline to production. All of the repositories representing pipeline stages can also be configured with different access permissions, to tailor the pipeline to your needs.

## 7.26.3 | Set Up OCI Clients To Work With Artifactory

JFrog Artifactory supports a variety of OCI clients, so you can find the best fit for your environment. The supported clients are:

- Podman
- BuildX
- BuildKit/ Buildctl
- WASM to OCI
- ORAS

### Note

While you can use Docker as an OCI client, it is not recommended, as the client will upload Docker media types instead of OCI.

### 7.26.3.1 | Configure Podman To Work With Artifactory

Podman (the Pod Manager) is a tool for managing, developing, and running OCI images on Linux systems. It is daemonless, which makes it more accessible and less vulnerable to security breaches than other clients.

You can configure Podman using two methods: using the `podman login` command, or manually updating your `~/.docker/config.json` file.

To configure Podman clients to work with Artifactory using the `podman login` command:

1. Login using the following Podman command:

### Note

Make sure to replace the placeholder in **bold** with your own JFrog host domain.

```
podman login <YOUR_JFROG_DOMAIN>
```

For example:

```
podman login my-awesome.jfrog.io
```

2. (Optional) Provide your Artifactory username and password or Identity Token. If anonymous access is enabled, you do not need to log in.

To manually set your credentials, copy the following snippet to your `~/.docker/config.json` file.

### Note

Make sure to replace the placeholders in **bold** with your own JFrog host domain, token, and email address.

```
{
  "auths": {
    "<YOUR_JFROG_DOMAIN>" : {
      "auth": "<Token>",
      "email": "<Email>"
    }
  }
}
```

For example:

```
{
  "auths": {
    "jfrog.org/artifactory" : {
      "auth": "RANDOM_TOKEN_YWRtaW46QVBVWjTEXkZTU4WT",
      "email": "johnfrog@email.com"
    }
  }
}
```

```
    }
}
```

### Push OCI Resources Using Podman

To push Podman images using Artifactory:

1. Tag your image using the following Podman command:

#### Note

Make sure to replace the placeholders in **bold** with your own JFrog host domain, desired repository path, image, and tag.

```
podman tag "<IMAGE_ID>" <YOUR_JFROG_DOMAIN>/<REPOSITORY>/<IMAGE>:<TAG>
```

For example:

```
podman tag "5ed3drf04b2be" my-awesome.jfrog.io/myproject-oci-remote/myociimage:latest
```

2. Push your image tag using the following Podman command:

#### Note

Make sure to replace the placeholders in **bold** with your own JFrog host domain, desired repository path, image, and tag.

#### Note

Force-set the image as OCI using the flag `-f=oci`

```
podman push <YOUR_JFROG_DOMAIN>/<REPOSITORY>/<IMAGE>:<TAG> -f=oci
```

For example:

```
podman push my-awesome.jfrog.io/myproject-oci-remote/myociimage:latest -f=oci
```

### Pull OCI Resources Using Podman

To pull Podman images using Artifactory, use the following Podman command:

#### Note

Make sure to replace the placeholders in **bold** with your own JFrog host domain, desired repository path, image, and tag.

```
podman pull <YOUR_JFROG_DOMAIN>/<REPOSITORY>/<IMAGE>:<TAG>
```

For example:

```
podman pull my-awesome.jfrog.io/myproject-oci-remote/myociimage:latest
```

### 7.26.3.2 | Configure BuildX To Work With Artifactory

BuildX is a Docker CLI plugin that allows support for multi-platform images and signed images, using the OCI specification. BuildX allows developers to build and push images for multiple platforms and architectures using a single command, which makes for a more effective and efficient development process.

To configure BuildX to work with Artifactory, log in using the following Docker command:

#### Note

Make sure to replace the placeholder in **bold** with your own JFrog host domain.

```
docker login <YOUR_JFROG_DOMAIN>
```

For example:

```
docker login my-awesome.jfrog.io
```

### Push OCI Resources Using BuildX

You can use the BuildX client to push multiple architecture or single architecture images.

To push OCI resources using Buildx:

1. Create and use a builder instance using the following Docker BuildX command:

```
docker buildx create --use
```

2. Build and push multiple images using the following command:

#### Note

Make sure to replace the placeholders in **bold** with your own architectures, JFrog host domain, desired repository path, image, and tag.

```
docker buildx build --platform <ARCHITECTURE_1>,<ARCHITECTURE_2> -t <YOUR_JFROG_DOMAIN>/<REPOSITORY>/<IMAGE>:<TAG> . --push
```

For example:

```
docker buildx build --platform linux/amd64,linux/arm64 -t my-awesome.jfrog.io/myproject-oci-remote/myociimage:latest . --push
```

### 7.26.3.3 | Configure BuildKit/ buildctl To Work With Artifactory

# JFrog Artifactory Documentation

## Displayed in the header

BuildKit/ buildctl is a toolkit that allows you to easily build multiarchitecture images.

To configure BuildKit to work with Artifactory:

1. Login using the following Docker command:

### Note

Make sure to replace the placeholder in **bold** with your own JFrog host domain.

```
docker login <YOUR_JFROG_DOMAIN>
```

For example:

```
docker login my-awesome.jfrog.io
```

2. Run your container locally using the following command:

### Note

Make sure to replace the placeholders in **bold** with your own container and image.

```
docker run --rm --privileged -d --name <CONTAINER> moby/buildkit
```

For example:

```
docker run --rm --privileged -d --name mybuildkit moby/buildkit
```

## Push OCI Resources Using BuildKit

To push and tag your images from the running container, use the following command:

### Note

Make sure to replace the placeholders in **bold** with your own architectures, JFrog host domain, desired repository path, image, and tag.

```
buildctl build \
--frontend=dockerfile.v0 \
--local context=. \
--local dockerfile=. \
--output type=image,name=<YOUR_JFROG_DOMAIN>/<REPOSITORY>/<IMAGE>:<TAG>,push=true \
--export-cache type=registry,ref=<YOUR_JFROG_DOMAIN>/<REPOSITORY>/<IMAGE>:<TAG>,mode=max,push=true \
--import-cache type=registry,ref=<YOUR_JFROG_DOMAIN>/<REPOSITORY>/<IMAGE>:<TAG>
```

For example:

```
buildctl build \
--frontend=dockerfile.v0 \
--local context=. \
--local dockerfile=. \
--output type=image,name=my-awesome.jfrog.io/myproject-oci-remote/myociimage:latest,push=true \
--export-cache type=registry,ref=my-awesome.jfrog.io/myproject-oci-remote/myociimage:latest,mode=max,push=true \
--import-cache type=registry,ref=my-awesome.jfrog.io/myproject-oci-remote/myociimage:latest
```

### 7.26.3.4 | Configure WASM to OCI To Work With Artifactory

ORAS is a client for working with OCI artifacts and registries.

To configure ORAS to work with Artifactory, log in using the following command:

### Note

Make sure to replace the placeholder in **bold** with your own JFrog host domain.

```
docker login <YOUR_JFROG_DOMAIN>
```

For example:

```
docker login my-awesome.jfrog.io
```

## Push OCI Resources Using WASM to OCI

To push WASM to OCI resources to Artifactory, use the following command:

### Note

Make sure to replace the placeholders in **bold** with your own WASM file, JFrog host domain, desired repository path, WASM file, and tag.

```
wasm-to-oci push <WASM_FILE> <YOUR_JFROG_DOMAIN>/<REPOSITORY>/<PACKAGE>:<TAG>
```

For example:

```
wasm-to-oci push mypackage.wasm my-awesome.jfrog.io/myproject-oci-remote/myociimage:latest
```

## Pull OCI Resources using WASM to OCI

To pull WASM to OCI resources from Artifactory, use the following command:

### Note

Make sure to replace the placeholders in **bold** with your own JFrog host domain, desired repository path, WASM file, and tag.

```
wasm-to-oci pull <YOUR_JFROG_DOMAIN>/<REPOSITORY>/<PACKAGE>:<TAG>
```

# JFrog Artifactory Documentation

## Displayed in the header

For example:

```
wasm-to-oci pull my-awesome.jfrog.io/myproject-oci-remote/myociimage:latest
```

### 7.26.3.5 | Configure ORAS To Work With Artifactory

ORAS is a client for working with OCI artifacts and registries.

To configure ORAS to work with Artifactory, log in using the following command:

#### Note

Make sure to replace the placeholder in **bold** with your own JFrog host domain.

```
oras login <YOUR_JFROG_DOMAIN>
```

For example:

```
oras login my-awesome.jfrog.io
```

Alternatively, you can provide your username and token in advance using the following command:

#### Note

Make sure to replace the placeholder in **bold** with your own JFrog host domain, username, and token.

```
oras login -u <USERNAME> -p <TOKEN> <YOUR_JFROG_DOMAIN>
```

For example:

```
oras login -u johnf -p cmVmddGtu0jE3NTkxNTSHV6MQ6T1RBakkyMj50ZjVRSEZTeVpxktJNmVJWlNy my-awesome.jfrog.io
```

### 7.26.4 | Use Referrers REST API to Discover OCI References

Starting from Artifactory version 7.90.1, Artifactory supports Referrers API, part of the OCI specification version 1.1 which provides a new way to link between images and their related information.

Using referrers API, you can connect an image to its signatures, SBOM certificates, attestations, Xray scan results, and other related artifacts, easily retrieve them together, and transfer them between repositories.

For more information, see [Open Container Initiative Distribution Specification](#).

#### How Referrers API Works in Artifactory

When deploying an image to a repository, you can use many different OCI-compliant tools to add referrers and attestations. This will create a new signature image in the folder of the original image- when you open the `manifest.json` file for that image, it will contain a 'subject' field that points to the original image. Using the Referrers API It also creates a `referrers.json` index file only visible to Admin users, which will be updated whenever this image is updated or new referrers are added.

Now these images are linked, and you can discover the connection via CLI or REST API.

To use the CLI, you can use functions like `oras discover` to find all associated artifacts, such as signatures, attestations, and SBOMs, that reference a specific image. For example, when running the following command:

```
oras discover <JFROG_PLATFORM_URL>/<REPOSITORY_NAME>/<IMAGE>:<DIGEST/ TAG>
```

You will get a list of all of the artifacts referencing your image.

To use REST API, use the following GET request:

```
GET <ARTIFACTORY_URL>/<REPOSITORY_NAME>/<IMAGE>/referrers/<DIGEST/ TAG>
```

To get a JSON file with a manifest list detailing all of the signatures associated with your image.

#### How to Use Referrers API

To enable referrers API on an image in an Artifactory repository:

1. Enable referrers API using the following feature flag in your system configuration file:

```
artifactory.oci.referrers.api.enabled=true
```

2. Select an image and a reference you would like to attach to it. Use the CLI command for the client you selected for attaching a signature. For example, to use Cosign to attach an SBOM certificate to an OCI image, use the following command:

#### Note

To use Cosign with OCI spec version 1.1, add the following feature flag to your system configuration file:

```
export COSIGN_EXPERIMENTAL=1
```

Also, make sure that you are using Cosign version 2.0.0 and above. For more information, see the [Cosign Changelog](#).

3. To verify that Referrers API is activated on an image, when pushing an image with a subject field, make sure that the response contains the following header with the SHA256 digest for the referenced image:

```
OCI-Subject: <DIGEST/ TAG>
```

### 7.26.5 | Limitations of OCI

# JFrog Artifactory Documentation

## Displayed in the header

It is currently possible to run Xray scans on container images that follow the OCI and Docker image specification only. Other content types wrapped in OCI such as WASM or any other payloads are not supported. For more information, see JFrog Xray.

### 7.27 | 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:

- The ability to provision ipk packages from Artifactory to an Opkg client from local and remote repositories.
- Calculation of Metadata for ipk packages hosted in Local Repositories .
- Access to remote Opkg resources (such as [downloads.openwrt.org](https://downloads.openwrt.org)) through Remote Repositories which provide the usual proxy and caching functionality.
- Providing GPG signatures that can be used by Opkg clients to verify packages.
- Complete management of GPG signatures using the Artifactory UI and the REST API.

#### Note

Artifactory signs repository metadata (not packages) for Opkg.

#### Integration Benefits

[JFrog Artifactory and Opkg Repositories](#)

#### 7.27.1 | Set Up an Opkg Repository

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.

This section contains the following topics:

- Set Up Local Opkg Repositories
  - Deploy an Opkg Package Using the UI
- Set Up Remote Opkg Repositories

Virtual repositories are not supported for Opkg.

##### 7.27.1.1 | Set Up Local Opkg Repositories

To create a new local repository that supports Opkg, in the **Administration** module, go to **Repositories| Repositories | Local** and set the **Package Type** to **Opkg**.

Artifactory supports the common Opkg index scheme which indexes each feed location according to all ipk packages in it.

##### 7.27.1.1.1 | Deploy an Opkg Package Using the UI

To deploy an Opkg package to Artifactory, go to the **Artifactory Repository Browser** and click the  **Deploy** icon.

Select your Opkg repository as the **Target** Repository, and upload the file you want to deploy.

Tip

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.

#### 7.27.1.2 | Set Up Remote Opkg 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, 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](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/](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.

#### 7.27.2 | Configure 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 libgpgme11-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) 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
```

### 7.27.3 | Sign Opkg Package Indexes

Artifactory uses your GPG public and private keys to sign and verify Opkg package indexes (note that Artifactory signs repository metadata, not packages).

To learn how to generate a GPG key pair and upload it to Artifactory, see [Managing Signing Keys](#).

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 in 0.3 versions

```
opkg-key add key.pub
```

#### Importing gpg keys to Opkg's keychain in 0.4 versions

```
mkdir -p /usr/etc/opkg/gpg
opkg-key add key.pub
cp -R /etc/opkg/gpg/* /usr/etc/opkg/gpg
```

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
```

#### Resolving Failed

If resolving fails with the following errors:

```
"opkg_verify_gpg_signature: No sufficiently trusted public keys found."
"pkg_src_verify: Signature verification failed for <repoName>."
```

One of the possible reasons can be that the trust level of the `key.pub` is not high enough, and should be upgraded.

### 7.27.4 | Opkg 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](#).

### 7.27.5 | REST API Support for Opkg

The Artifactory REST API provides extensive support for signing keys and recalculating the repository index as follows:

- Set GPG Public Key
- Get GPG Public Key
- Set GPG Private Key
- Set GPG Pass Phrase
- Recalculate the index

### 7.28 | P2 Repositories

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](#).

#### Tip

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.

#### Integration Benefits

[JFrog Artifactory and Eclipse P2 Repositories](#)

### 7.28.1 | Set Up a P2 Repository

# JFrog Artifactory Documentation

## Displayed in the header

To use P2 repositories, follow the steps below:

- Define a Virtual P2 Repository
- Select Local Repositories to Add to Your Virtual P2 Repository
- Select Remote Repositories to Add to Your Virtual P2 Repository
- Create the P2 Repositories
- Configure Eclipse to Work With Your P2 Virtual Repository

Local repositories are not supported for P2.

### 7.28.1.1 | Define a Virtual P2 Repository

- From the Administration module, select **Repositories** | **Repositories** | **Virtual** and Create a new virtual repository and set **P2** as the **Package Type**.

#### Tip

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.

### 7.28.1.2 | Select Local Repositories to Add to Your Virtual P2 Repository

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 Suffix** 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 **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!/`

### 7.28.1.3 | Select Remote Repositories to Add to Your Virtual P2 Repository

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.

A common example is the main Juno repository: <http://download.eclipse.org/releases/juno>

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).

### Tip

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!/?](http://eclipse.org/equinox-sdk.zip!/)

#### 7.28.1.4 | Create the P2 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:

Action	Description
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

#### 7.28.1.5 | Configure Eclipse to Work With Your P2 Virtual Repository

You are now ready to configure eclipse to work with the virtual repository you have created above.

1. In the Eclipse menu, select **Help | Install new Software** and then click **Add**.
2. In the **Add Repository** popup, enter a name for your repository (we recommend using the same name used in Artifactory) and its URL:
  
  
  
3. Eclipse will then query Artifactory for available packages and update the screen to display them as below:

#### 7.28.2 | Allow Anonymous Access for P2

Artifactory supports P2 repositories with **Allow Anonymous Access** enabled. For more information, see [Allow Anonymous Access](#).

When **Allow Anonymous Access** is enabled, Artifactory will not query the P2 client for authentication parameters by default, so you need to indicate to Artifactory to request authentication parameters differently.

You can override the default behavior by selecting the **Force Authentication** checkbox in the New or Edit Repository dialog.

When set, Artifactory will first request authentication parameters from the P2 client before trying to access this repository.

#### 7.28.3 | Integrate P2 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's pom.xml file as displayed in the snippet below:

```
<repository>
  <id>eclipse-indigo</id>
  <layout>p2</layout>
  <url>http://localhost:8081/artifactory/p2-virtual</url>
</repository>
```

### Note

The P2 virtual repository should contain URLs to all local repositories with an optional sub-path in them where Tycho build artifacts reside.

#### 7.28.4 | Use Multiple P2 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.

...but in fact, the virtual repository only really includes one remote repository

#### 7.28.5 | Configure a Remote P2 Repository for GWT

The Google Plugin for Eclipse does not support Eclipse 4.7 (Oxygen) or later and is no longer available. For more details please refer to the Google Cloud Platform documentation.

To configure your P2 repository to proxy GWT, set the remote repository URL in Artifactory to be: <http://storage.googleapis.com/gwt-eclipse-plugin/v3/release>

## 7.29 | PHP Composer Repositories

Artifactory supports PHP Composer repositories on top its existing support for advanced artifact management.

Artifactory support for Composer provides:

- Provisioning Composer packages from Artifactory to the Composer command line tool from all repository types.
- Calculation of metadata for Composer packages hosted in Artifactory local repositories.
- Access to remote Composer metadata repositories (Packagist and Artifactory Composer repositories) and package repositories (such as Github, Bitbucket etc..) through remote repositories [Remote Repositories](#) which provide proxy and caching functionality.
- Assigning access privileges according to projects or development teams.
- Support PHP Composer versions 1 and 2 (virtual repositories are only supported for Composer version 2 only),
- Support for downloading Drupal file version 7 and 8 from remote repositories.

### Integration Benefits JFrog Artifactory and PHP Composer Repositories

#### 7.29.1 | Set Up a PHP Composer Repository

You can set up the following repository types:

- Local Composer repositories
- Remote Composer repositories
- Virtual Composer repositories

##### 7.29.1.1 | Set Up Local Composer Repositories

# JFrog Artifactory Documentation

## Displayed in the header

To enable calculation of Composer package metadata, from the **Administration** Module, go to **Repositories | Repositories | Local** and set **PHP Composer** to be the **Package Type** when you create your local Composer repository.

### Composer v2 Support

Starting with release 7.24.1, Artifactory supports PHP Composer V2, which features faster download times and enhanced performance. As a result, the **Enable Composer V1 Indexing** option is disabled by default for new repositories, but you can still enable V1 indexing if required. (Existing Composer repositories created before release 7.24.1 continue to use V1 indexing as before.) Bear in mind that a change to this setting requires a full reindexing.

#### 7.29.1.1.1 | Deploy 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 the Artifactory 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.

### 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

#### 7.29.1.2 | Set Up Remote Composer 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:

- From the **Administration** module, go to **Repositories | Repositories | Remote** and set **PHP Composer** to be its **Package Type**
- Set the **Repository Key**, and enter the repository URL (e.g. `https://github.com/`) in the **URL** field as displayed below .

### Note

When pointing to Packagist (`https://asset-packagist.org`), Artifactory will only resolve Composer packages. npm and Bower packages should be pulled using dedicated repositories for those package types.

- In the **Composer Settings** section, select **GitHub** as the **Git Provider**, and leave the default **Registry URL** (e.g. `https://packagist.org/`).
- Finally, click **Save & Finish**.

## URL vs. Registry URL

To avoid confusion:

The **URL** is the URL of your Git provider where the actual package binaries are hosted.

The **Registry URL** refers to the URL where the package index files holds the hosted metadata.

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://jfrog-art.com/artifactory/api/composer/composer-local>.

### 7.29.1.2.1 | Set Composer Remote Repositories to Work With Drupal 7 and 8 Package Type

From Artifactory version 7.24.1 (SaaS users) and higher supports Composer V2 allowing download of Drupal version 7 and 8 files from remote repositories.

To define a remote repository to proxy Drupal files as well as the public Composer Packagist repository follow these steps.

1. From the **Administration** module, go to **Repositories | Repositories | Remote | Basic tab** and set **PHP Composer** to be its **Package Type**.
2. Set the URL to <https://ftp.drupal.org/>.
3. Under the **PHP Composer Settings** section, set the **Registry URL** field to <https://packages.drupal.org/>

### 7.29.1.3 | Set Up Virtual Composer Repositories

A Virtual Repository defined in Artifactory aggregates PHP packages from both local and remote repositories that are included in the virtual repositories. 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 PHP Composer repository follow these steps:

1. Create a new Virtual Repository, in the **Administration** module, under **Repositories | Repositories | Virtual**, click **New Virtual Repository** and set **PHP Composer** as the **Package Type**.
2. Set the **Repository Key** value.
3. Select the underlying local and remote PHP Composer repositories to include under the **Repositories** section.
4. You can optionally also configure your **Default Deployment Repository**.

You can set the following in the Advanced tab.

Field	Description
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.

#### 7.29.2 | Use the Composer Command Line

Once the Composer client is installed, you can access Composer repositories in Artifactory through its command line interface.

# JFrog Artifactory Documentation

## Displayed in the header

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 Cloud, the URL would be:

```
https://<server name>.jfrog.io/artifactory/api/composer/<repository key>
```

Once you have created a Composer repository, from the **Application** module go to **Artifactory |Artifacts | Artifact Repository Browser** and click **Set Me Up** to get code snippets you can use to set your Composer repository URL in your `config.json` file.

### Composer config.json file

Windows: `%userprofile%\composer\config.json`

Linux: `~/.composer/config.json`

### Replace 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": [
    ...
  ]
}
```

### GitHub Authentication

From November 2020, GitHub no longer accepts account passwords when authenticating via the REST API and instead requires the use of token-based authentication such as a personal access token for all authenticated API operations on [GitHub.com](#).

#### 7.29.3 | Clean 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 Package. 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

#### 7.29.4 | View Individual Composer Package Information

# JFrog Artifactory Documentation

## Displayed in the header

Artifactory lets you view selected metadata of a Composer package directly from the UI.

From the **Application** module, go to **Artifacts | Tree** and drill down to select the package archive file you want to inspect. The metadata is displayed in the **Composer Info** tab.

## 7.30 | Pub Repositories

From JFrog Artifactory 7.31.10, the Pub repository is supported for the Dart programming language, which contains reusable libraries & packages for Flutter, Angular Dart, and general Dart programs. This gives you full control of your deployment and resolving of Pub packages. Pub downloads your Dart package's dependencies, compiles your packages, makes distributable packages, and uploads them to pub.dev, the Dart community's package registry. You can contribute to this book on GitHub.

### About Dart Programming Language

Dart is an Open Source, client-side programming language developed by Google, which is designed for client development, such as web and mobile apps. Dart is an object-oriented, class-based, garbage-collected language with a C-style syntax, and can also be used to build server and desktop applications.

Flutter is an Open-Source UI SDK also developed by Google. It allows the development of iOS/Android apps and uses Dart as its programming language.

Pub repositories in Artifactory offer the following benefits:

- Secure and private local Pub repositories with fine-grained access control
- The ability to serve as a proxy for remote Pub resources and to cache downloaded Pub packages to keep you independent of the network and the remote resource
- Metadata calculation of the Pub packages hosted in the Artifactory local repositories
- Version management: Archiving older versions of the packages uploaded to local repositories
- Source and binary management

### Supported Pub Version

Artifactory supports Pub version 2.15.0-268.8.beta and above.

#### 7.30.1 | Pub Repository Structure

The Pub repository structure is as follows.

```
|-.pub/<packageName>.json  
|---<packageName>/<packageName>-<version>.tar.gz
```

Note the following:

- The index of the packages will be populated under the .pub , each package has its own json file that index the package versions.
- The binary of the library is populated under the package folder with a convention of the package name and the package version , for example: pedantic/pedantic-1.9.1.tar.gz.

### Deployment Structure

All deployment of Pub packages into Artifactory must be under the <PACKAGE\_NAME>/<PACKAGE\_NAME>-<VERSION>.tar.gz structure.

If packages are not deployed under this structure, they will not be included in any index file.

#### 7.30.2 | Set Up a Pub Repository

You can set up the following repository types:

- Local Pub Repositories
- Remote Pub Repositories
- Virtual Pub Repositories

Follow the steps according to each repository type below. A Pub package (`tar.gz`) is deployed to a local PUB repository, and resolved using all repository types.

You can download packages from a local, remote, or virtual Pub repository.

### 7.30.2.1 | Set Up a Local Pub Repository

Local repositories enable you to deploy pub (`tar.gz`) packages. Artifactory calculates the metadata for all packages and indexes them to allow users to download these packages through the Pub client.

To create a Pub local repository, navigate to the **Administration** module, go to **Repositories| Repositories | Local | New Local Repository** and select **Pub** as the **Package Type**.

### 7.30.2.2 | Set Up a Remote Pub Repository

Remote repositories enable you to proxy and cache Pub packages.

To specify that a Remote Repository supports Pub packages, you need to set its **Package Type** to **Pub** when it is created.

### 7.30.2.3 | Set Up a Virtual Pub Repository

A Virtual Repository defined in Artifactory aggregates packages from both local and remote repositories.

This allows you to access both locally hosted Pub packages and remote proxied Pub repositories from a single URL defined for the Virtual repository.

To define a virtual Pub repository, do the following:

1. Create a Virtual Repository, and set the **Package Type** to **Pub**.
2. Select the underlying local and remote Pub repositories to include in the **Basic** settings tab.

### 7.30.2.4 | Pub SemVer 2.0 Package Support

Artifactory requires applying SemVer 2.0 rules for Pub repositories, which means you can now use pre-release numbers with dot notation or add metadata to the version, for example: `MyApp.3.0.0-build.60`, `MyApp.1.0+git.52406`.

Artifactory serves the requests for downloading packages using SemVer 2.0 rules. For example, if the latest version for a certain package is in a SemVer 2.0 convention, Artifactory will return it to the client. NuGet packages with the SemVer 2.0 convention are served from local, remote, and virtual repositories and for both Pub API v2, and v3 feeds.

### 7.30.3 | Configure the Pub Client to Work With Artifactory

To use Artifactory with your Pub client, you will first need to set Artifactory as a Pub repository, and then to resolve and deploy the relevant Dart/ Flutter package.

#### Prerequisite

You will need to generate an authentication token. For more information, see Access Token Authorization Headers.

#### Step 1: Add Artifactory to your / etc/pub/repositories File

1. Navigate to **Application Module** | **Artifactory** | **Artifacts**.
2. Select the desired repository.
3. Select **Set Me Up**.
4. In the **Configure** tab, add the repository to your client using the following command and run it.

#### HTTPS-Mode Only

Pub authentication to Artifactory is supported only through HTTPS-Only mode.

## Step 2: Deploy Dart/ Flutter Packages

To deploy a Dart/ Flutter package into an Artifactory repository, use the following cURL with the relevant path parameters:

### cURL

```
curl -u<USERNAME>:<PASSWORD> -T <file>.tar.gz " https://localhost:8080/artifactory/dart-pub-local/<>/<PACKAGE_NAME>/<FILE>/<VERSION>.tar.gz"
```

Deploying a Package Using the UI

### Indexing Dart/Flutter Packages

For your files to be indexed properly, it is very important to ensure that all deployment of Dart/ Flutter packages into Artifactory occurs under the <REPOSITORY>/<PACKAGE\_NAME>/<FILE>/<VERSION> structure. Packages deployed anywhere else will not be indexed.

To deploy a Dart/ Flutter package to Artifactory, do the following:

1. Navigate to **Artifactory | Artifacts | Deploy**.
2. Select your Pub repository as the **Target Repository**.
3. In the **Target Path**, specify the relative path in the target repository.

### Step 3: Resolve and Deploy Dart/ Flutter Packages

To resolve Dart packages:

1. In the Application module, navigate to **Artifactory | Artifacts**.
2. In the Artifact Tree Browser, select a Pub repository and click **Set Me Up**.

#### 7.30.4 | [View Individual Pub Package Information](#)

Artifactory lets you view selected metadata of a Pub package directly from the UI.

In the **Artifact Repository Browser**, select your local Pub repository and scroll down to find and select the package you want to inspect.

The metadata is displayed in the **Pub Info** tab, or viewed in the **Packages** view.

#### 7.30.5 | Re-index a Pub Repository

You can trigger an asynchronous re-indexing of a local Pub repository either through the UI or using the REST API.

In the **Artifact Tree Browser**, select your Pub repository, right-click and select **Recalculate Index** from the list (requires Admin privileges).

To re-index a Pub repository through the REST API, see Calculate Pub Repository Metadata.

#### 7.30.6 | REST API Support for Pub

The Artifactory REST API enables the recalculation of the repository index, as described in Calculate Pub Repository Metadata.

### 7.31 | Puppet Repositories

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.
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 the Remote Repositories which provide 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.

#### 7.31.1 | Set Up a Puppet Repository

You can set up the following repository types:

- Local Puppet repositories
- Remote Puppet repositories
- Virtual Puppet repositories

##### 7.31.1.1 | Set Up Local Puppet Repositories

To enable the calculation of Puppet module metadata in local repositories, from the **Administration** module go to **Repositories | Repositories | Local** set the **Package Type** to **Puppet** when you create the repository.

### 7.31.1.1 | Use Puppet 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.

### 7.31.1.2 | Set Up Remote Puppet Repositories

A **Remote Repositories** 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 **Administration** module, under **Repositories | Repositories | Remote**, click **New Remote Repository**.
2. In the New Remote 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**.

### 7.31.1.3 | Set Up Virtual Puppet Repositories

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, from the **Administration** module, select **Repositories | Repositories | Virtual**, 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.

i

#### 7.31.2 | Use 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 Cloud the URL would be:

`https://<server name>.jfrog.io/artifactory/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 **Application Module | Artifactory | Artifacts** Tree Browser and click the **Set Me Up** button to get helpful 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.

#### Replacing the default repository

To replace the default repository with a URL pointing to a Puppet repository in Artifactory, add the 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)
```

#### 7.31.3 | Use 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
```

#### 7.31.4 | Use r10k for Puppet

`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
```

### 7.31.5 | Deploy Modules with Puppet Publish

To deploy modules using puppet publish:

1. Set your Puppet credentials
2. Deploy your puppet modules

#### 7.31.5.1 | Set Your Puppet 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:8080/artifactory/api/puppet/puppet-local
```

#### Note

Make sure you have an Artifactory user in order to publish modules.

### 7.31.5.2 | Deploy Your Puppet 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 the Puppet module. Select your module(s), and click **Deploy**.

#### 2. Using Artifactory REST API

For Example:

```
curl -uadmin:AP7eCk6M6JokQpjXbkGytt7r4sf -PUT http://localhost:8080/artifactory/puppet-local/<TARGET_FILE_PATH> -T <PATH_TO_FILE>
```

### 7.31.6 | Work with Artifactory and Puppet without Anonymous Access

By default, Artifactory allows anonymous access to Puppet repositories. This is defined in the **Administration** 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.

### 7.31.7 | Use Puppet Search

Artifactory supports a variety of ways to search for artifacts. For details, please refer to [Searching for Artifacts\\_OLD](#). 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.

### Tip

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.

### 7.31.8 | Clean Up the Local Puppet Cache

The Puppet client saves caches of downloaded modules and 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>.

### 7.31.9 | View 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.

### 7.31.10 | Use 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 that 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;
    }
}
```

# JFrog Artifactory Documentation

## Displayed in the header

```
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://10.6.125:8080 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 (.*)
    ## NOTE: The 'REQUEST_SCHEME' Header is supported only from apache version 2.4 and above
    RewriteCond %{REQUEST_SCHEME} (.*)
    RewriteRule (.*)
    RewriteCond %{HTTP_HOST} (.*)
    RewriteRule (.*)
    ##

    RewriteRule ^/$ /artifactory/webapp/ [R,L]
    RewriteRule ^/artifactory(/)?$ /artifactory/webapp/ [R,L]
    RewriteRule ^/artifactory/webapp$ /artifactory/webapp/ [R,L]

    RequestHeader set Host %{my_custom_host}
    RequestHeader set X-Forwarded-Port %{my_server_port}
    ## NOTE: {my_scheme} requires a module which is supported only from apache version 2.4 and above
    RequestHeader set X-Forwarded-Proto %{my_scheme}
    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 (.*)
    ## NOTE: The 'REQUEST_SCHEME' Header is supported only from apache version 2.4 and above
    RewriteCond %{REQUEST_SCHEME} (.*)
    RewriteRule (.*)
    RewriteCond %{HTTP_HOST} (.*)

    RewriteCond %{HTTP_HOST} (.*)
```

# JFrog Artifactory Documentation

## Displayed in the header

```
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}
RequestHeader set X-Forwarded-Port %{my_server_port}
## NOTE: {my_scheme} requires a module which is supported only from apache version 2.4 and above
RequestHeader set X-Forwarded-Proto %{my_scheme}
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://10.6.125:8080 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/$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}
    RequestHeader set X-Forwarded-Port %{my_server_port}
    ## NOTE: {my_scheme} requires a module which is supported only from apache version 2.4 and above
    RequestHeader set X-Forwarded-Proto %{my_scheme}
    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
```

### 7.31.11 | REST API Support for Puppet

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.
- Get Puppet Module: Returns information about a specific Puppet module.
- Get Puppet Releases: Returns a list of all Puppet releases hosted by the specified repository.
- Get Puppet Release: Returns information about the specific Puppet module's release.

### 7.32 | PyPI Repositories

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.org/>) through **Remote Repositories** which provides 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.

#### PyPI Supported Clients

In Artifactory, PyPI repositories support the following clients:

- Poetry version 1.2.0 and later
- Twine version 3.3.0 and later
- Pip version 20.2 and later

#### 7.32.1 | Set Up PyPI Repositories on Artifactory

Artifactory supports the following repository types:

- Local PyPI Repositories
- Remote PyPI Repositories
- Virtual PyPI Repositories

##### 7.32.1.1 | Set Up Local PyPI Repositories

To create a new PyPI local repository:

1. From the **Administration** module select **Repositories**.
2. Click **Create a Repository** and select **Local** from the dropdown.
3. Select **PyPI** as the Package Type.



4. Click **Create Local Repository**.

**Read More:** [JFrog Artifactory and PyPI Repositories](#)

##### 7.32.1.2 | Set Up Remote PyPI Repositories

A remote repository defined in Artifactory serves as a caching proxy for a registry managed at a remote URL such as <https://pypi.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.

# JFrog Artifactory Documentation

## Displayed in the header

To create a repository to proxy a remote PyPI repository follow the steps below:

1. From the **Administration** module select **Repositories**.

In the **Administration** module under **Repositories | Repositories | Remote**, and click **New Remote Repository**.

2. Click **Create a Repository** and select **Remote** from the dropdown.

3. Select **PyPI** as the Package Type.

4. Enter the **Repository Key** value.

5. The URL and Registry URL settings depend on whether you are proxying the public external PyPI repository, or a PyPI repository hosted on another Artifactory server.

**For a public, external PyPI repository:** Change the **URL** field to <https://files.pythonhosted.org/>, and set the **Registry URL** field to <https://pypi.org/>.

**For a PyPI repository hosted on another Artifactory instance:** Set the remote repository's PyPI API URL in the **Registry URL** field. For example, to proxy a PyPI repository called "python-project" hosted by an Artifactory instance at <https://<YOUR-JFROG-URL>/artifactory/>, set the **URL** and **Registry URL** to

- **URL:**<https://<YOUR-JFROG-URL>/artifactory/python-project>
- **Registry URL:**<https://<YOUR-JFROG-URL>/artifactory/api/pypi/python-project>

as shown below:

#### PyPI remote repository URL

You should not include /simple in the PyPI **remote repository URL**. This suffix is 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 .

6. Click **Create Remote Repository**.

#### Remote Artifactory

If the remote repository is also managed by an Artifactory server, then you need to point to its PyPI API URL in the **Registry URL** field. For example, <http://my.remote.artifactory/api/pypi/python-project>

##### 7.32.1.3 | Set Up Virtual PyPI 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:

1. From the **Administration** module, go to **Repositories** .
2. Click **Create a Repository** and select **Virtual** from the dropdown.
3. Select **PyPI** as the Package Type.
4. Select the underlying local and remote PyPI repositories to include in the **Basic** settings tab, and click **Create Virtual Repository**.

### 7.32.2 | Set up PyPI Clients to Work with Artifactory

You can configure your PyPI repositories with the following clients:

- Poetry (from version 1.2.0 and later)
- Twine (from version 3.3.0 and later)
- Pip (from version 20.2 and later)

#### 7.32.2.1 | Set Up PyPI with a Poetry Client

The process for setting up PyPI with a Poetry client is done in 3 stages:

- Configure the Poetry client
- Publish a Python package to Artifactory
- Install a Python Package

##### Configure the Poetry Client

1. In the command line, configure the upload endpoint for your publishable repository with the following commands:

```
poetry config repositories.artifactory-<REPOSITORY_NAME>  
https://<YOUR_JFROG_DOMAIN>/artifactory/api/pypi/<REPOSITORY_KEY>
```

```
poetry config http-basic.artifactory-<REPOSITORY_NAME> <USER> <PASSWORD/TOKEN>
```

For example:

```
poetry config repositories.artifactory-pypi-local  
https://my-awesome.jfrog.io/artifactory/api/pypi/pypi-local  
poetry config http-basic.artifactory-pypi-local johnfrog RANDOM_TOKEN_YWRtaW46QVBBWJjTExkZTU4WT
```

2. Add the repository source with the source add command as follows:

```
poetry config http-basic.<REPOSITORY_NAME> <USER><PASSWORD/TOKEN>  
poetry source add <REPOSITORY_NAME> https://<YOUR_JFROG_DOMAIN>/artifactory/api/pypi/<REPOSITORY_NAME>/simple
```

For example:

```
poetry config http-basic.pypi-local johnfrog RANDOM_TOKEN_YWRtaW46QVBBWJjTExkZTU4WT  
poetry source add pypi-local https://my-awesome.jfrog.io/artifactory/api/pypi/pypi-local/simple
```

##### Note

The source add command inserts the URL into the project's `pyproject.toml`.

##### Publish a Python Package to Artifactory

To publish a Python package to Artifactory, run the following command:

```
poetry publish --repository artifactory-<REPOSITORY_NAME> --dist-dir dist
```

For example:

```
poetry publish --repository artifactory-pypi-local --dist-dir dist
```

##### Install a Python Package

To install a Python package, run the following command:

```
poetry add --source <REPOSITORY_NAME> <PACKAGE_NAME>
```

For example:

```
poetry add --source pypi-local my_pypi_package
```

### 7.32.2.2 | Set Up PyPI with a Twine Client

The process for setting up PyPI with a Twine client is done in 2 stages:

1. Configure the Twine client
2. Upload a Python package to Artifactory

#### Configure the Twine Client

To deploy packages using Twine, add an Artifactory repository to the `.pypirc` file (usually located in your home directory) as follows:

```
[distutils]
index-servers = <REPOSITORY_NAME>
[<REPO_NAME>]
repository: https://<YOUR_JFROG_DOMAIN>/artifactory/api/pypi/<REPOSITORY_NAME>
username: <USER>
password: <PASSWORD/TOKEN>
```

For example:

```
[distutils]
index-servers = pypi-local
[pypi-local]
repository: https://my-awesome.jfrog.io/artifactory/api/pypi/pypi-local
username: johnfrog
password: RANDOM_TOKEN_YWRtaW46QVBVWjzTExkZTU4WT
```

#### Upload a Python Package to Artifactory

To upload a Python package to Artifactory, use the following command:

```
twine upload --repository <REPOSITORY_NAME> <PATH_TO_FILE>
```

For example:

```
twine upload --repository pypi-local ~/workspace/my_file.tar.gz
```

### 7.32.2.3 | Set Up PyPI with a Pip Client

The process for setting up PyPI with a Pip client is done in 2 stages:

1. Configure the Pip client
2. Install a Python package to Artifactory

#### Running on Windows

To use Artifactory PyPI repositories on Windows, make sure to set the required environment variables for Python and Pip.

Note that on Windows platforms, `%HOME%\pip\pip.ini` replaces the `pip.conf` file described in the sections below and should be reachable through your HOME path.

#### Configure the Pip Client

To configure your Pip client to work with Artifactory, add the following to `~/.pip/pip.conf`:

```
[global]
index-url = https://<USER>:<PASSWORD/TOKEN>:<YOUR_JFROG_DOMAIN>/artifactory/api/pypi/<REPO_NAME>/simple
```

For example:

```
[global]
index-url = https://johnfrog:RANDOM_TOKEN_YWRtaW46QVBVWjzTExkZTU4WT:my-awesome.jfrog.io/artifactory/api/pypi/pypi-local/simple
```

#### Note

If credentials are required they should be embedded in the URL.

#### Install a Python Package to Artifactory

To install packages using pip, run:

```
pip3 install <PACKAGE_NAME>
```

For example:

```
pip3 install my_pypi_package
```

### 7.32.3 | Publish PyPI Packages 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.

### 7.32.4 | Search for PyPI Packages

#### Use Pip Search

Artifactory supports search using *pip's search command* in local and virtual repositories. For example:

```
$ 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** and **irbench** are two locally installed packages in different local repositories, and both repositories are aggregated by the **pypi-virtual** repository.

#### Note

Since `pip search` has been deprecated, Artifactory no longer supports pip search in remote repositories, including smart repositories. As such, if a virtual repository includes remote repositories, those remote repositories will not be scanned for search results.

#### 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.

#### Use 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.

### 7.32.5 | View Metadata of PyPI Packages

Artifactory lets you view selected metadata for 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.

### 7.32.6 | Work with PyPI Remote Repositories with the Custom Registry Suffix

You can set a custom suffix instead of the default simple like in cases of DevPi.

To set the `devpi` registry suffix to the server suffix:

Use the root URL in the URL and Registry URL. For example: `http://m.devpi.net`.

In order to search, include the required scope in the index URL (as in the devpi example, it could be root/pypi).

```
$ pip search frog-fu --index http://localhost:8081/artifactory/api/pypi/devpi/root/pypi/
```

To install, include the desired scope in the index url (like in devpi example, it could be root/pypi)

```
$ pip install frog-bar -i http://localhost:8081/artifactory/api/pypi/devpi/root/pypi/simple
```

#### 7.32.7 | Watch the PyPI Screencast

Watch this video to learn more about setting up Artifactory as a PyPI repository:



#### 7.32.8 | Limitations of PyPI Package Registry

- As Artifactory supports PyPI legacy, both underscores and dots are allowed in package names. However, they will not be aggregated into one package: for example, jfrog.pypi and also jfrog\_pypi will continue to be two separate packages, with the same behavior as in PyPI registry.

#### 7.32.9 | Best Practices for Working with PyPI Repositories

The following topics are recommended best practices when working with PyPI repositories:

- Resolve from Artifactory Using Pip
- Publish PyPI Packages to Artifactory

##### 7.32.9.1 | Resolve 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.

##### Using a Valid SSL Certificate with pip and Artifactory

pip uses packages from the local cache, (i.e. from the machine on which the pip client is running on), only if the download URL of the package is a trusted host with a valid SSL certificate. This means that if your Artifactory instance is not running with a valid SSL certificate, requests for packages will always first reach Artifactory even if the packages exist on the local cache.

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**.

7.32.9.1.1 | [Specify the PyPI Repository on the Command Line](#)

#### Index URL

When accessing a PyPI repository through Artifactory, the repository URL should 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 Cloud, the URL would be:

`https://<server name>.jfrog.io/artifactory/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
```

#### Note

The default configuration snippet for Nginx and Apache using reverse proxy contains the `X-JFrog-Override-Base-Url` by default. If reverse proxy is not used in your environment, you need to add the header manually to the request, for example:

```
-H "X-JFrog-Override-Base-Url: http://$ART_HOST"
```

Instead of adding the header manually, you can also run the request on port 8081, or add a slash (/) at the end of the request:

```
http://$ART_HOST/artifactory/api/pypi/pypi-virtual/simple/
```

7.32.9.1.2 | [Use PyPI 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.

7.32.9.1.3 | [Use a Pip 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](#).

# JFrog Artifactory Documentation

## Displayed in the header

### 7.32.9.1.4 | Use a Pip 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
```

### 7.32.9.2 | Publish PyPI Packages to Artifactory

#### Limitation

Users should not deploy packages in Artifactory to repositories in the paths `pypi-repo/packages/**` or `pypi-repo/simple/**`, as these are saved namespaces used by Artifactory for internal logic. If you attempt to deploy to these paths, Artifactory returns a 400 error.

There are several ways to publish PyPI packages to Artifactory:

- Use `distutils` or `setuptools`
- Use `.netrc`
- Upload Authenticated PyPI Packages to JFrog Artifactory
- Upload PyPI Egg and Wheel Packages

### 7.32.9.2.1 | Use PyPI 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 instruction were written for Python 2.7 and `setuptools` in mind. Using different version of Python, or different tools such `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.

### 7.32.9.2.1.1 | 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.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.

### 7.32.9.2.2 | Use .netrc to Upload PyPI Packages

Uploading PyPI packages to Artifactory using a `.netrc` file credentials is supported. For more information on the `.netrc` file , see [The .netrc file and Purpose of the '.netrc' file](#).

#### Note

- For information on how to use a Keyring provider via PyPI repositories, click [here](#).
- The Twine client is not compatible with `.netrc` credentials.

To use the `.netrc` file:

1. To create a `.netrc` file in your root directory and set it with the minimum required permissions (Chmod 600), if you do not have one already, run the following command:

```
touch ~/.netrc
```

# JFrog Artifactory Documentation

## Displayed in the header

2. Run the following command to populate the `.netrc` file with your Artifactory URL and credentials according to the `.netrc` file format:

### Note

Make sure to replace the placeholders in **bold** with your own Artifactory URL, username, and password or identity token.

```
echo "machine <ARTIFACTORY_URL> login <USERNAME> password <PASSWORD/TOKEN>" > ~/.netrc
```

### 7.32.9.2.3 | Upload Authenticated PyPI Packages to JFrog Artifactory

To upload authenticated PyPI packages to JFrog Artifactory:

1. Create the `.pypirc` file using the following code, and add your access token from JFrog Artifactory for the username `pypiadmin..`

```
[distutils]
index-servers =
    private-repository

[private-repository]
repository = https://soleng.jfrog.io/artifactory/api/pypi/demo-pypi-local
username = pypiadmin
password =
eyJ2ZXIiOiyIiwidHlwIjoiSl0UiwiYWxnIjoiUlMyNTiLCJraWQiOijSypadzNjUu13WBBSWNRa01RRjN0dla2Yml5M3dwCXdRQ0txUkxLaXhRIn0..eyJleHQiOij7XCJyZXvY2FibGvCijpcInRydWVcIn0iLCJzdW
ii01JqZmFj0DAxZTlycTMza3ljhNHQxMwtangybmwemo1KC91c2yc1vcHlwawFkbwluiwiC2NwIjoiYXBwbG1lZC1wZXJtaXNzaW9uc1wYWRtaW4iLCJhdWQioiIqQcoiLCJpc3MiOijqZmZ1QDawMCisimV4CcI6MTY4
OTY5ODA5MywiaWF0IjoxNjU4MTYyMDkzLCJqdGkiOijZwy1YWY5Ni0zNTkyLTrjOTQtYWYyNs1kZmIxY2UZGU4YzIifQ.NI50Xpw0Nh7Asd3f_sY3tMyzM-2_07c3WyWEpbJrDPx08eKoLRp10vGEF8J03HyRQ0H7Ybf2-
Cn8wf9FMoUG1gxGfm7_yc24xwVLCINjg0B2ASyRSvActwdTzwgVPvEMUqCPSCU4_SGgg6061IDxxImRfgZlwFn-
whF08b8dCV7EV4dXGvH7iVb33W2JguE9KQFP7KKQlaEr06pGNuPfox1jb1lH0oRpAPzvykda2i6Q2q3ZCobJZ9Rp8NqZYQfEm42YtIa0vA1E5fGepZgdjzHaalCztJDHo-
BWjMiDFP0LRTHAS07F52t8p3vsZhw5Neov_trFrQ
```

2. Copy the `.pypirc` file to your Users folder.

- For macOS users, create the file under `%USERPROFILE%`
- For Windows users, create the file under `c:\users\<name>`

3. Validate that the `setup.py` file contains all of the following fields. It should be automatically created, but if it is not, add the file to your root folder with the following content.

```
#!/usr/bin/env python

from setuptools import setup

setup(
    name='demo-python-example',
    version='1.0',
    description='Project example for building Python project with JFrog products',
    author='JFrog',
    author_email='jfrogsupport@jfrog.com',
    packages=['helloworld'],
    install_requires=['PyYAML>3.11', 'nltk'],
)
```

4. Create a PyPI package.

```
py setup.py sdist bdist_wheel
```

The following displays an example output.

```
C:\Users\johnk\helloworld>py setup.py sdist bdist_wheel
running sdist
running egg_info
writing demo_python_example.egg-info\PKG-INFO
writing dependency_links to demo_python_example.egg-info\dependency_links.txt
writing requirements to demo_python_example.egg-info\requirements.txt
writing top-level names to demo_python_example.egg-info\top_level.txt
reading manifest file 'demo_python_example.egg-info\SOURCES.txt'
writing manifest file 'demo_python_example.egg-info\SOURCES.txt'
warning: sdist: standard file not found: should have one of README, README.rst, README.txt, README.md
running check
creating demo-python-example-1.0
creating demo-python-example-1.0\demo_python_example.egg-info
creating demo-python-example-1.0\helloworld
copying files to demo-python-example-1.0...
copying setup.py -> demo-python-example-1.0
copying demo_python_example.egg-info\PKG-INFO -> demo-python-example-1.0\demo_python_example.egg-info
copying demo_python_example.egg-info\SOURCES.txt -> demo-python-example-1.0\demo_python_example.egg-info
copying demo_python_example.egg-info\dependency_links.txt -> demo-python-example-1.0\demo_python_example.egg-info
copying demo_python_example.egg-info\requirements.txt -> demo-python-example-1.0\demo_python_example.egg-info
copying demo_python_example.egg-info\top_level.txt -> demo-python-example-1.0\demo_python_example.egg-info
copying helloworld\app.py -> demo-python-example-1.0\helloworld
Writing demo-python-example-1.0\setup.cfg
Creating tar archive
removing 'demo-python-example-1.0' (and everything under it)
running bdist_wheel
running build
running build_py
C:\Users\johnk\installs\lib\site-packages\setuptools\command\install.py:34: SetuptoolsDeprecationWarning: setup.py install is deprecated. Use build and pip and other
standards-based tools.
  warnings.warn(
installing to build\bdist.win-amd64\wheel
running install
running install_lib
creating build\bdist.win-amd64\wheel
creating build\bdist.win-amd64\wheel\helloworld
```

# JFrog Artifactory Documentation

## Displayed in the header

```
copying build\lib\helloworld\app.py -> build\bdist.win-amd64\wheel.\helloworld
running install_egg_info
Copying demo_python_example.egg-info to build\bdist.win-amd64\wheel.\demo_python_example-1.0-py3.10.egg-info
running install_scripts
creating build\bdist.win-amd64\wheel\demo_python_example-1.0.dist-info\WHEEL
creating 'dist\demo_python_example-1.0-py3-none-any.whl' and adding 'build\bdist.win-amd64\wheel' to it
adding 'helloworld/app.py'
adding 'demo_python_example-1.0.dist-info\METADATA'
adding 'demo_python_example-1.0.dist-info\WHEEL'
adding 'demo_python_example-1.0.dist-info\top_level.txt'
adding 'demo_python_example-1.0.dist-info\RECORD'
removing build\bdist.win-amd64\wheel
C:\Users\johnk\helloworld>
```

5. Upload the package to JFrog Artifactory.

```
py setup.py sdist upload -r private-repository
```

The following displays an example output.

```
C:\Users\johnk\helloworld>py setup.py sdist bdist_wheel
running sdist
running egg_info
writing demo_python_example.egg-info\PKG-INFO
writing dependency_links to demo_python_example.egg-info\dependency_links.txt
writing requirements to demo_python_example.egg-info\requirements.txt
writing top-level names to demo_python_example.egg-info\top_level.txt
reading manifest file 'demo_python_example.egg-info\SOURCES.txt'
writing manifest file 'demo_python_example.egg-info\SOURCES.txt'
warning: sdist: standard file not found: should have one of README, README.rst, README.txt, README.md
running check
creating demo-python-example-1.0
creating demo-python-example-1.0\demo_python_example.egg-info
creating demo-python-example-1.0\helloworld
copying files to demo-python-example-1.0...
copying setup.py -> demo-python-example-1.0
copying demo_python_example.egg-info\PKG-INFO -> demo-python-example-1.0\demo_python_example.egg-info
copying demo_python_example.egg-info\SOURCES.txt -> demo-python-example-1.0\demo_python_example.egg-info
copying demo_python_example.egg-info\dependency_links.txt -> demo-python-example-1.0\demo_python_example.egg-info
copying demo_python_example.egg-info\requirements.txt -> demo-python-example-1.0\demo_python_example.egg-info
copying demo_python_example.egg-info\top_level.txt -> demo-python-example-1.0\demo_python_example.egg-info
copying helloworld\app.py -> demo-python-example-1.0\helloworld
Writing demo-python-example-1.0\setup.cfg
Creating tar archive
removing 'demo-python-example-1.0' (and everything under it)
running bdist_wheel
running build
running build_py
C:\Users\johnk\lib\site-packages\setuptools\command\install.py:34: SetuptoolsDeprecationWarning: setup.py install is deprecated. Use build and pip and other standards-based tools.
    warnings.warn(
installing to build\bdist.win-amd64\wheel
running install
running install_lib
creating build\bdist.win-amd64\wheel
creating build\bdist.win-amd64\wheel\helloworld
copying build\lib\helloworld\app.py -> build\bdist.win-amd64\wheel.\helloworld
running install_egg_info
Copying demo_python_example.egg-info to build\bdist.win-amd64\wheel.\demo_python_example-1.0-py3.10.egg-info
running install_scripts
creating build\bdist.win-amd64\wheel\demo_python_example-1.0.dist-info\WHEEL
creating 'dist\demo_python_example-1.0-py3-none-any.whl' and adding 'build\bdist.win-amd64\wheel' to it
adding 'helloworld/app.py'
adding 'demo_python_example-1.0.dist-info\METADATA'
adding 'demo_python_example-1.0.dist-info\WHEEL'
adding 'demo_python_example-1.0.dist-info\top_level.txt'
adding 'demo_python_example-1.0.dist-info\RECORD'
removing build\bdist.win-amd64\wheel
C:\Users\johnk\helloworld>
```

6. Your package was uploaded successfully to the JFrog Platform. Check it out by navigating to **Application > Artifactory > Artifacts** and searching for the package.

#### 7.32.9.2.4 | Upload PyPI Egg and Wheel Packages

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.org/pypi` if no repository is specified.

#### The 'register' command should be omitted

When uploading directly to `pypi.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**.

#### 7.32.9.2.5 | Use PyPI Enforce Layout

In Artifactory versions 7.77 and above, you can use the Enforce Layout feature to bring Artifactory closer in alignment with the PyPi.org registry.

This feature helps you keep a consistent naming method for PyPI packages and reduce package name duplications due to a mismatch between package name and metadata. We recommend using this feature alongside File Path Name Normalization: for more information, see [Use File Path Name Normalization](#) and [Using Both File Path Name Normalization and Enforce Layout](#).

#### Backward Compatibility

Enforce Layout applies to all PyPI repositories in your Artifactory instance. Previously uploaded packages that already exist in your instance will not be updated but if you attempt to upload a newer version or overwrite an existing one you will be prevented from doing so since the file path name sections must match the metadata values.

This feature allows you to upload only on artifacts where the file name and version in the artifact path match the package name and version stored in the artifact's metadata.

The following examples explain how this feature applies without File Path Name Normalization:

To enable PyPI Enforce Layout, set the system property:

```
pypi.enforce.layout = true
```

Artifactory will return an error when trying to perform actions on an artifact with a name that does not match the package metadata, with a description of the issue:

Error	Description
403	Action prevented due to Enforce Layout policy - package file name/version <ARTIFACT_NAME> does not match the " + package internal name/version <ARTIFACT_NAME> in the metadata file
403	Action prevented due to Enforce Layout policy - could not find metadata file to compare

#### 7.32.9.2.6 | [Use PyPI File Path Name Normalization](#)

In Artifactory versions 7.77 and above, you can use the File Path Name Normalization feature to bring Artifactory closer in alignment with the PyPI.org registry.

This feature helps you keep a consistent naming method for PyPI artifacts to reduce potential package name duplications due to the use of non-normalized characters.

We recommend using this feature alongside Enforce Layout: for more information, see [Use Enforce Layout](#) and [Using Both File Path Name Normalization and Enforce Layout](#).

#### Backward Compatibility

File Path Name Normalization applies to all PyPI repositories in your Artifactory instance, previously uploaded packages that already exist in your instance will not be updated but if you attempt to upload a newer version or overwrite an existing one you will be prevented from doing so since the file path name does not conform to the normalization rules.

When this feature is enabled, Artifactory blocks the deployment of files with path name/distribution values that do not conform to the PyPI standard naming conventions, as specified in the Python documentation pages [Source Distribution File Name](#) and [File Format](#).

```
{NAME}-{VERSION}-{OPTIONAL-PARAMETERS}.tar.gz  
{DISTRIBUTION}-{VERSION}-{OPTIONAL-PARAMETERS}.whl
```

# JFrog Artifactory Documentation

## Displayed in the header

The most common example of how this feature provides value is the prevention of Dash (-) that is considered a Reserved Character between the different segments of the naming path (name/distribution, version, optional parameters, etc.) from being used within the name/distribution segment.

For example, see whether different names would be allowed when File Path Name Normalization is enabled:

Encrypted Assertion

```
pypi.enforce.naming.normalization = true
```

Artifactory will return an error when trying to perform actions on an artifact whose name does not match the naming conventions, with a description of the issue:

Error	Description
403	Action prevented due to artifact file name normalization requirements - artifact name <ARTIFACT_NAME> does not conform to normalized requirements

7.32.9.2.7 | [Using Both PyPI File Path Naming Normalization and Enforce Layout](#)

**File Path Name Normalization and Enforce Layout** work both independently and together to create a safety net for correct package naming.

The Enforce Layout setting only verifies that the file path and metadata have matching properties, while the File Path Name Normalization setting verifies that the names align with the PyPI naming conventions. Using both will ensure that you do not have duplicate or redundant packages.

Please note that although PyPI allows the use of the dash in the internal metadata names, our enforce layout feature is strict and your metadata names must match the file path name always.

For example, this will be the behavior when both File Path Name Normalization and Enforce Layout are enabled:

## 7.33 | RPM Repositories

Artifactory is a fully-fledged RPM repository. As such, it enables:

- RPM metadata calculation for RPMs hosted in Artifactory local repositories.
- Deploying RPM Modules to Artifactory local repositories.
- Provisioning RPMs directly from Artifactory to YUM clients.
- Detailed RPM metadata views from the web UI.
- Providing GPG signatures that can be used by the YUM client to authenticate RPMs.

### Also valid for YUM

The instructions on this page can be used for RPM repositories and YUM repositories interchangeably

#### 7.33.1 | Generate 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
primary.xml.gz	Contains an XML file describing the primary metadata of each RPM archive.
filelists.xml.gz	Contains an XML file describing all the files contained within each RPM archive.
other.xml.gz	Contains an XML file describing miscellaneous information regarding each RPM archive.
repomd.xml	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.

##### 7.33.2 | Trigger 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 Updates

RPM metadata is automatically calculated:

1. When deploying, removing, copying, and moving an RPM file.
2. When performing content import (both system and repository imports).

###### Manual Updates

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.

###### Note

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.

###### Index 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.

##### 7.33.3 | Set Up an RPM Repository

To create an RPM local repository, in the **Administration** module go to **Repositories** | **Repositories** | **Local**, click **New Local Repository** and select **RPM** as the **Package Type**.

###### 7.33.3.1 | Set Up Local RPM 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**.

Field	Description
RPM Metadata Folder Depth	<p>Informs Artifactory under which level of directory to search for RPMs and save the repodata 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 RPMs and saves the repodata directory at \$REPO-KEY/repoadata.</p> <p>Using a different depth is useful in cases where generating metadata for a repository separates its artifacts by name, version and architecture. This will allow you to create multiple RPM repositories under the same Artifactory RPM repository.</p> <p>For example:</p> <p>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 Depth to 1 and the repodata directory is saved at REPO_ROOT/ARTIFACT_NAME/repoadata :</p> <pre>REPO_ROOT/\$ARTIFACT_NAME/\$ARTIFACT_VERSION/\$ARCHITECTURE/FILE_NAME - or - rpm-local/foo/1.0/x64/foo-1.0-x64.rpm</pre>
<b>Note</b>	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 depth.depth.
Auto-calculate RPM Metadata	When set, RPM metadata calculation is automatically triggered by the actions described above.
Enable File List Indexing	When set, RPM metadata calculation will also include indexing the filelists.xml metadata file.
RPM Group File Names	A comma-separated list of YUM group files associated with your RPM packages.  Note that at each level (depth), the repodata directory in your repository may contain a different group file name, however each repodata directory may contain only 1 group metadata file (multiple groups should be listed as different tags inside the XML file. For more details, see <a href="#">YUM Documentation</a> ).

#### 7.33.3.2 | Set Up Remote RPM Repositories

Artifactory remote repositories support RPMs out-of-the-box, and there 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 **Administration** module under **Repositories** | **Repositories** | **Remote**, click **New Remote Repository** to create a new remote repository.
2. Set the **Repository Key**, and specify the URL to the remote registry in the **URL** field. When using a Yum client, use the URL of the folder containing your /repodata folder- for example, [http://mirror.centos.org/centos/7/os/x86\\_64/](http://mirror.centos.org/centos/7/os/x86_64/).

**Note**

Running `docker pull centos:latest` will return CentOS 8, which has reached End of Life. To use CentOS, pull a supported version: for a list of supported versions, see [CentOS Mirror](#). If you need to use CentOS 8, you can do so by changing the URL in the URL field to <https://vault.centos.org/>.

3. Click **Save & Finish**.
4. Back in the **Application | Artifactory | 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.

**Note**

For CentOS 8 users, edit or create the following files with root privileges:

- `sudo vi /etc/yum.repos.d/CentOS-Linux-AppStream.repo`
- `sudo vi /etc/yum.repos.d/CentOS-Linux-BaseOS.repo`

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
```

### 7.33.3.3 | Set Up Virtual RPM Repositories

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, from the **Administration** module, go to **Repositories | Repositories | Virtual**, set the **Package Type** to be **RPM**, and select the underlying local and remote RPM repositories to include in the **Basic** settings tab.

#### Note

When using virtual RPM repositories, Artifactory does not support using YUM or DNF commands that use the updateinfo.xml metadata file.

To allow deploying packages to this repository, set the [Default Deployment Repository](#).

### 7.33.3.4 | Deploy RPM Modules to a Local Repository

You can deploy RPM modules to the repodata folder using the modules.yaml file which contains all the desired modules and then proceed to trigger a repository index to update the repomd.xml and other metadata files with the modules information.

1. You can perform the upload directly in the JFrog Platform UI or using the following command.

```
curl -u <USERNAME>:<PASSWORD> -XPUT "http://localhost:8081/artifactory/rpm-local/<PATH_TO_REPODATA_FOLDER>/modules.yaml -T <TARGET_MODULES_FILE_PATH>"
```

2. Trigger the reindex by running the Calculate YUM Repository Metadata API.

### 7.33.4 | Sign RPM Metadata

Artifactory supports using a GPG key to sign RPM metadata repositories (not packages) for authentication by the YUM client.

To generate a pair of GPG keys and upload them to Artifactory, see [Managing Signing Keys](#).

### 7.33.5 | Install RPM Packages Using Yum

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>
```

### Managing Weak Dependencies in Yum/RPM with Recommends Tags

#### Note

This feature is supported from release version Artifactory 7.106.2 Cloud

In RPM-based package management systems like Yum, packages can specify optional dependencies using the Recommends tag in their metadata. These dependencies are not required for the core functionality of the package but are recommended for enhanced performance or features. By default, the yum client installs these recommended dependencies automatically when installing a package, unless explicitly disabled.

You can manage how Yum handles weak (recommended) dependencies by configuring the Recommends tags in the `primary.xml` metadata of RPM repositories.

#### Feature Flag Control: Enabling/Disabling Recommends Tags

The inclusion of Recommends tags in `primary.xml` is now configurable via a feature flag `yum.local.install.recommended.dependencies.enabled`

When enabled, Yum will install any recommended dependencies listed in the `primary.xml` metadata of RPM local repositories. When disabled, the `recommends` XML tag is not added to repository metadata.

#### Default Behavior for Installing Recommended Dependencies

When you install a package with Yum, it will by default install any weak dependencies (i.e., recommended packages) listed for that package. For example:

```
yum install -y simple-package
```

This command will install the simple package and any recommended packages that are listed as dependencies.

#### Preventing Installation of Weak Dependencies

If you want to install a package without its weak dependencies (without installing recommended packages), you can disable this behavior in one of the following ways:

- **Disable via Command Line**

To prevent installing recommended packages when installing a package, use the `--setopt=install_weak_deps=False` flag in the yum command. For example:

```
yum --setopt=install_weak_deps=False install -y simple-package
```

This command installs simple-package but skips the installation of any recommended dependencies.

- **Disable via Configuration File**

You can permanently configure Yum to skip weak dependencies by modifying its configuration. There are two main ways to do this:

- **Modify the Main Yum Configuration File (`/etc/yum.conf`):**

Open the Yum configuration file and add the following setting under the `[main]` section:

```
[main]
install_weak_deps=False
```

This change will apply to all Yum operations globally.

- **Create a Custom Configuration File in `/etc/yum/yum.conf.d/`:**

Alternatively, you can create a specific configuration file in the `/etc/yum/yum.conf.d/` directory to disable weak dependencies for specific use cases. For example, create a new file called `disable-weak-deps.conf`:

```
/etc/yum/yum.conf.d/disable-weak-deps.conf
```

Then, add the following configuration in the `disable-weak-deps.conf` file.

```
[main]
install_weak_deps=False
```

This allows you to keep different configurations in separate files for different scenarios or environments.

### 7.33.6 | 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> -PUT 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.

### 7.33.7 | Manage 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
```

# JFrog Artifactory Documentation

## Displayed in the header

### 7.33.7.1 | Attach a YUM Group to a Repository

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.

### 7.33.7.2 | Use YUM Group Commands

The following table lists some useful YUM group commands:

Command	Description
<code>yum groupinstall &lt;Group ID&gt;</code>	Install the YUM group. The group must be deployed to the root of the YUM local repository.
<code>yum groupremove &lt;Group ID&gt;</code>	Remove the RPM group
<code>yum groupupdate &lt;Group ID&gt;</code>	Update the RPM group. The group must be deployed to the root of the YUM local repository.
<code>yum groupinfo &lt;Group ID&gt;</code>	List the RPM packages within the group.
<code>yum grouplist   more</code>	List the YUM groups

### 7.33.7.3 | Set YUM Group Properties

YUM group properties can be set in the `/etc/yum.config` file as follows:

Setting	Allowed values	Description
<code>overwrite_groups</code>	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.
<code>groupremove_leaf_only</code>	0 or 1	Determines YUM's behavior when the <code>groupremove</code> 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.
<code>enable_group_conditionals</code>	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.
<code>group_package_types</code>	optional, default, mandatory	Tells YUM which type of packages in groups will be installed when <code>groupinstall</code> is called. Default is: default, mandatory

### 7.33.8 | Set Up Yum Authentication

This section includes the following guides:

- Configure Proxy Server Settings for YUM
- Configure SSL Settings for YUM

# JFrog Artifactory Documentation

## Displayed in the header

### 7.33.8.1 | Configure Proxy Server Settings for YUM

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>
```

### 7.33.8.2 | Configure SSL Settings for Yum

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.

### 7.33.9 | Use 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]
```

### 7.33.10 | View 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.

## 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 see [Property Sets](#).

### 7.33.11 | Watch the RPM Screencast

Watch this short screencast to learn how easy it is to host RPMs in Artifactory.



### 7.34 | RubyGems Repositories

# JFrog Artifactory Documentation

## Displayed in the header

### Breaking Change: RubyGems Dependencies REST API

RubyGems is deprecating their support of the /dependencies REST API, and JFrog is removing the support for local repositories to align with RubyGems. For more information, please read this blog post by RubyGems: [RubyGems.org Dependency API Deprecation](#) or this Knowledge Base article by JFrog: [RubyGems.org Local Repositories Dependency API Deprecation](#).

This feature will reach end-of-life on June 1st, 2024.

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
- Support for Bundler Compact index for Local and Remote repositories, providing you with the latest version of the package that is compatible with your installed Ruby version of the project.

### Limitations of RubyGems in Artifactory

Priority resolution is currently not supported for RubyGems virtual repositories following the deprecation of the /dependencies REST API (as the Artifactory default indexing is currently using compact index).

#### 7.34.1 | Set Up a RubyGems Repository

##### 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 Cloud, the URL would be:

`https://<server name>.jfrog.io/artifactory/api/gems/<repository key>`

You can create the following repository types:

- Local repositories
- Remote Repositories
- Virtual Repositories

#### 7.34.1.1 | Set Up Local RubyGems 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 **Administration** module, under **Repositories | Repositories | Local**, click **New Local Repository** and set **Gems** 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**.

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/
```

7.34.1.1.1 | [Use Local RubyGems Repositories](#)

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 key>/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 **Application** module | **Artifactory** | **Artifacts** | **Gems-local** and click **Set Me Up**.

Alternatively you could use the `gem push` command with `--host`, and optionally, `--key` to specify the relevant API key.

**Note**

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.

7.34.1.1.2 | [Local RubyGems Repositories Default Files](#)

Upon creation of a Local RubyGems repository, Artifactory populates it with the following set of files that are required for indexing the repository:

- `rubygems-update-*.gem` (created under the `gems` folder of the repository)
- `quick/Marshal.*`
- `latest_specs.*.gz`
- `prerelease_specs.*.gz`
- `specs.*.gz`

The local RubyGems repository is displayed in the [Application module](#) | [Artifactory](#) | [Artifacts](#).

7.34.1.2 | [Set Up Remote RubyGems 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 [Administration](#) module, under [Repositories](#) | [Repositories](#) | [Remote](#), click [New Remote Repository](#) and set **Gems** to be the **Package Type**.
2. Set the **Repository Key**, and specify the **URL** to the remote repository. The example below references [rubygems.org](http://rubygems.org).

7.34.1.2.1 | [Use Remote RubyGems Repositories](#)

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 Application module | Artifactory | Artifacts Tree Browser, select your newly created repository and click **Set Me Up**.
2. Copy the source URL from the **RubyGems Sources** section.

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 /...`

7.34.1.3 | [Set Up Virtual RubyGems Repositories](#)

# JFrog Artifactory Documentation

## Displayed in the header

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.

### 7.34.1.3.1 | Use Virtual RubyGems Repositories

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!)`

### 7.34.2 | Use the REST API for RubyGems

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/&lt;repository key&gt;/api/v1/gems/my_gem.(json yaml)</code>	Optionally indicate JSON / YAML (default: JSON)
Search	<code>curl http://localhost:8081/artifactory/api/gems/&lt;repository key&gt;/api/v1/search.(json yaml)?query=[query]</code>	Will search for gems with name containing <code>query</code>
Dependencies	<code>curl http://localhost:8081/artifactory/api/gems/&lt;repository key&gt;/api/v1/dependencies?gems=[gem1,...]</code>	Use a csv of gem names for the value of <code>gems</code>
Yank	<code>curl -X DELETE http://localhost:8081/artifactory/api/gems/&lt;repository key&gt;/api/v1/yank -d 'gem_name=gem_name' -d 'version=0.0.1' -d 'platform=ruby'</code>	<b>Deletes</b> 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/&lt;repository key&gt;/reindex</code>	Re-creates all spec index files.

REST Command	Curl Syntax Example	Remarks
Update index	<code>curl -X POST http://localhost:8081/artifactory/api/gems/&lt;repository key&gt;/updateIndex</code>	Updates all spec index files if needed.

#### 7.34.3 | View 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.

#### 7.34.4 | Retrieve the Latest RubyGems Package Compatible With Your Ruby Versions

The Bundler Compact Index feature allows you to retrieve the latest RubyGems version compatible with your installed Ruby version in the project applies to local, remote, and virtual repositories.

To apply this feature, set the `artifactory.gems.compact.index.enabled=true` value in the `artifactory.system.properties` file.

### 7.35 | SBT Repositories

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.

#### 7.35.1 | Set Up an SBT Repository

You can set up the following repository types:

- Local SBT repositories
- Remote SBT repositories
- Virtual SBT repositories

##### 7.35.1.1 | Set Up Local SBT 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, from the **Administration** module, under **Repositories | Repositories | Local**, set the **Package Type** to **SBT**.

7.35.1.2 | Set Up Remote SBT 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 **Administration** module, under **Repositories | Repositories | Remote**, click **New Remote Repository**.
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 [Remote Repositories](#).

7.35.1.3 | Set Up Virtual SBT 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, from the **Administration** module, go to **Repositories | Repositories | Virtual**, set the **Package Type** to SBT, and select the underlying local and remote SBT repositories to include in the **Basic** settings tab.

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](#).

#### 7.35.2 | Configure SBT Client To Work With Artifactory

To configure SBT to resolve and deploy artifacts through SBT repositories defined in Artifactory, simply select one of the SBT repositories in the **Application** module, go to [Artifactory | Artifacts | Artifact Repository Browser](#) and click **Set Me Up**. Artifactory will display code snippets you can use in the relevant SBT files.

##### 7.35.2.1 | Configure SBT 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 (`C:\Users\%USERNAME%\sbt\repositories` on Windows).

# JFrog Artifactory Documentation

## Displayed in the header

```
[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 ( C:\Program Files (x86)\sbt\conf\sbtopts on Windows)

For more details on SBT proxy repositories, please refer to [Proxy Repositories in the SBT Reference Manual](#).

### 7.35.2.2 | Configure SBT 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

### 7.35.3 | Deploy SBT 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> is the host URL and port on which Artifactory is running, and repo-key is the Artifactory repository to which you are deploying artifacts.

### 7.35.4 | Fork Sample SBT Project

A sample SBT project that uses Artifactory is available on GitHub and can be freely forked.

## 7.36 | Swift Registry

From JFrog Artifactory version 7.41.4, Artifactory provides full support for managing Swift packages. Aggregating multiple Swift registries under a virtual repository Artifactory provides access to all your Swift packages through a single URL for both upload and download.

### Swift Version Support

Artifactory supports Swift version 5.7 and above.

### Did you know?

**Swift** is a general-purpose programming language built using a modern approach to safety, performance, and software design patterns.

The **Swift Package Manager**: A tool for managing the distribution of Swift code. It's integrated with the Swift build system to automate the process of downloading, compiling, and linking dependencies.

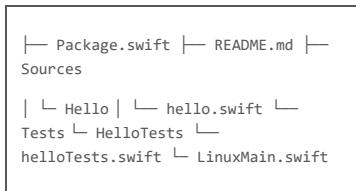
**Swift repository (Catalog)**: <https://swiftpackageregistry.com/> is a catalog of Git repositories, each containing a Swift package. The Swift package manager is an open-source project for managing the distribution of source code, aimed at making it easy to share code and reuse others' code. The tool compiles and links Swift packages, managing dependencies, versioning, and supporting flexible distribution and collaboration models. For more information, see [Swift Package Manager](#).

As a fully-fledged Swift registry on top of its capabilities for advanced artifact management, Artifactory's support for **Swift** provides:

- Calculation of Metadata for Swift packages hosted in Artifactory's local repositories.
- Access to remote registries through [Remote Repositories](#) which provide the usual proxy and caching functionality.
- The ability to access multiple Swift registries from a single URL by aggregating them under a [Virtual Repository](#). This overcomes the limitation of the Swift client which can only access a single registry at a time.
- Compatibility with the Swift command line tool to deploy and remove packages and more.
- Support for validating remote Swift repository metadata.

### 7.36.1 | Swift Repository Structure

The Swift package structure is as follows.



### Swift Deployment Structure

All deployment of Swift packages into Artifactory can be deployed under the following structures:

- <SCOPE>/<NAME>/<NAME>-<VERSION>.zip structure.
- <scope>/<name>/<version>: Based on the following Swift Publish API.

Note that the packages have to be deployed according to this structure, otherwise they will not be included in the index file.

### 7.36.2 | Set Up a Swift Registry

You can set up the following repository types:

- Local Repositories
- Remote Repositories
- Virtual Repositories

Follow the steps according to each repository type below.

You can download packages from a local, remote, or virtual Swift registry.

#### 7.36.2.1 | Set Up a Local Swift Repository

Local repositories enable you to deploy Swift (.swift) packages. Artifactory calculates the metadata for all packages and indexes them to allow users to download these packages through the Swift client.

Artifactory allows you to define any layout for your Swift registries. In order to upload packages according to your custom layout, you need to package your Swift files using the Swift source archive. This creates the .zip file for your package which you can then upload to any path within your local Swift repository.

To create a Swift local repository:

1. Navigate to the [Administration](#) module, and go to [Repositories | Repositories | Local | New Local Repository](#).
2. Select **Swift** as the **Package Type**.

#### 7.36.2.2 | Set Up a Remote Swift Repository

##### Note

Swift Remote repositories are currently only supported as [Smart Remote repositories](#). Artifactory is the only Swift registry available today, so you would not be able to proxy other remote resources.

Please note that Swift remote repositories can only connect to registries that utilize the swift protocol. URLs such as [swiftpackageregistry.com](#) are considered HTML catalogs, since they do not host any actual packages and simply point to a third-party Git repository.

A Remote Repository defined in Artifactory serves as a caching proxy for a registry managed at a remote URL. Artifacts (such as .zip 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 swift registry.

Remote Repositories enable you to proxy and cache Swift packages.

To define a remote repository to proxy a remote swift registry:

1. In the **Administration** module, under **Repositories | Repositories | Remote**, click **New Remote Repository**.
2. On the New Remote Repository page:
  - a. Set the Package Type to **Swift** and the Repository Key value.
  - b. Specify the URL to the remote registry in the URL field.
  - c. Click **Save & Finish**.

#### 7.36.2.3 | Set Up a Virtual Swift Repository

A Virtual Repository defined in Artifactory aggregates packages from both local and remote repositories.

This allows you to access both locally hosted Swift packages and remote proxied Swift repositories from a single URL defined for the Virtual Repository.

To define a virtual Swift repository, do the following:

1. In the **Administration** module, under **Repositories | Repositories | Virtual**, click **New Virtual Repository**.
2. In the New Virtual Repository dialog, set the Package Type to **Swift**.
3. Select the underlying local and remote Swift registries to include in the Basic settings tab.
4. Click **Save & Finish** to create the repository.

#### 7.36.3 | Configure the Swift Client to Work Opposite Artifactory

To use Artifactory with your Swift CLI, you must generate an access token. Then you can proceed to resolve and deploy the relevant Swift package.

1. Navigate to **Application Module | Artifactory | Artifacts**.
2. Select the desired repository.
3. Select **Set Me Up** and follow the instructions. You can log in to your client using one of these methods:

# JFrog Artifactory Documentation

## Displayed in the header

- Log in with your username and password.
- Log in using an authentication token in your Swift login command.

### Note

From Artifactory version 7.55.1, Swift 5.8 is supported with authentication support. For more information, see the [Swift Documentation](#).

To log in using an authentication token:

1. To set your Swift registry, run the following command:

```
swift package-registry set --global https://example.com/artifactory/api/swift/swift-local
```

2. To login to the registry using Swift, run the following command:

```
swift package-registry login https://example-registry.com \
--username jappleseed \
--password alpine \
```

This will create the `.swiftpm/configuration/registries.json` file as:

```
{
  "registries": {
    "[default)": {
      "url": "https://example-registry.com"
    }
  },
  "authentication": {
    "example-registry.com": {
      "type": "basic",
      "loginAPIPath": "artifactory/api/swift/swift-local"
    }
  },
  "version": 1
}
```

Starting from Swift version 5.8, you can use the new Swift login command. For more information, see the [Swift Documentation](#).

To use the new Swift login:

1. To set your Swift registry, run the following command:

```
swift package-registry set --global https://example.com/artifactory/api/swift/swift-local
```

2. To login to the registry using Swift, run the following command:

```
swift package-registry login https://example-registry.com \
--username jappleseed \
--password alpine \
```

3. This will create the `.swiftpm/configuration/registries.json` file as:

```
{
  "registries": {
    "[default)": {
      "url": "https://example-registry.com"
    }
  },
  "authentication": {
    "example-registry.com": {
      "type": "basic"
      "loginAPIPath": "artifactory/api/swift/swift-local"
    }
  },
  "version": 1
}
```

### 7.36.4 | Configure the Swift Client to Work With HTTP

By default, the Swift Client only works with HTTPS protocol. If you attempt to configure the client with an HTTP URL, you will get the following error:

```
Error: invalid URL:<URL>
```

To modify the Swift client to work with the HTTP protocol, do the following:

1. To set Artifactory as a Swift repository, run the following command using an HTTPS URL:

```
swift package-registry set <url>
```

This creates the `.swiftpm/configuration/registries.json` file, which contains the following information:

```
{
  "registries" : {
    "[default]" : {
      "url" : "<url to RT>/artifactory/api/swift/swift-local"
    }
  },
  "version" : 1
}
```

2. Modify the URL in the `<url to RT>` field in the `.swiftpm/configuration/registries.json` file to HTTP instead of HTTPS.

### 7.36.5 | Search for Swift Packages

You can search for Swift Packages, using the [Artifact Package Search](#).

### 7.36.6 | Re-Index a Swift Repository

You can trigger an asynchronous re-indexing of a local Swift repository either through the UI or using the REST API.

This will also reindex the git index and, as a result, will also index the remote repositories.

In the [Artifact Tree Browser](#), select your Swift repository, right-click and select **Recalculate Index** from the list. Requires Admin privileges.

To reindex a Swift repository through the REST API refer to the following REST API: Calculate Swift Index

## 7.37 | Terraform/ OpenTofu Repositories

From JFrog Artifactory 7.38.4, JFrog provides a fully-fledged Terraform repository solution giving you full control of your deployment and resolving process of Terraform Modules, Providers, and Backend packages.

Starting from Artifactory version 7.81.0, Jfrog also supports integration with OpenTofu to manage Terraform packages. To configure Artifactory with OpenTofu, see [Configure Terraform Provider Registry with Artifactory Using OpenTofu](#) and [Configure Terraform Backend Repository with Artifactory Using OpenTofu](#). After the initial configuration, all documentation instructions for Terraform also apply to OpenTofu.

### Did you know?

Terraform is an infrastructure as code (IaC) tool that allows developers to build, change, and version infrastructure safely and efficiently. Terraform is written in HCL language. Code in the Terraform language is stored in plain text files with the .tf file extension. For more information, see the [Terraform Language](#).

The Terraform Registry in the JFrog Platform offers the following benefits:

- Secure and private local Terraform Modules registry
- Secure and private local Terraform Providers registry
- Proxy remote Terraform Module and Provider resources with caching to keep you independent of the network and the remote resource.
- Virtual Terraform repositories that support a single URL through which to manage the resolution and deployment of all your Terraform Modules and Providers.

The Terraform Backend Repository in the JFrog Platform offers the following benefits:

- A Remote State Storage Provider
- Support for multiple Workspaces
- Built-in Secure State Encryption storage
- Comprehensive State snapshot history
- State content viewer with advanced search abilities

### Terraform CLI Version Support

Artifactory supports Terraform CLI version 1.0.0 and above.

### 7.37.1 | Supported Terraform Repository Types

The JFrog Terraform repository solution supports several Terraform repository types and dedicated configurations, all customized to accommodate the Terraform environment-specific settings and requirements.

The following table describes the Terraform repository implementation in the JFrog Platform.

Terraform Repository Type	Terraform Type	Local Repository	Remote Repository	Virtual Repository
Terraform	Module/ Provider	Select the type from the <b>Terraform Registry Type</b> list in the Basic tab	No differentiation between Modules and Providers	No differentiation between Modules and Providers
Terraform BE	Backend	Dedicated repository type	Not available	Not available

### 7.37.1.1 | Recommended Terraform Repository Setup

It is recommended to set up your Terraform repositories according to the following Terraform repository structure in the JFrog Platform to fully benefit from the full-fledged Terraform solution:

- Local Terraform Module Registry
- Local Terraform Provider Registry
- Remote Terraform Registry
- Virtual Terraform Registry: Aggregates the Local Terraform Module and the Local Terraform Provider registry
- Local Terraform Backend (BE) Repository.

### 7.37.2 | Get Started with Terraform

1. Create dedicated Module and Provider registries.
2. Create a Terraform Backend Repository.

### 7.37.3 | Terraform Registry

The Terraform registry in Artifactory allows you to create dedicated repositories for each of the following unique Terraform components:

- **Providers:** A set of plugins that interact with cloud providers, SaaS providers, and other APIs.
- **Modules:** Serve as containers for multiple resources that are used together. Modules contain a collection of .tf files kept together in a directory.

The Modules and Providers have different settings for local repositories but are the same when it comes to configuring remote and virtual repositories.

To learn more about the Terraform repository solution in the JFrog Platform, see [Terraform Repositories](#).

#### 7.37.3.1 | Use Terraform Module Registries

Terraform Modules are Terraform configurations that can be called and configured by other configurations. They serve as containers for multiple resources that are used together. They contain a collection of .tf files kept together in a directory. To create a module, pack all your .tf files into a Zip archive and deploy to the Terraform local repository. For more information, see [Creating Terraform Modules](#).

##### Terraform Module Repository Structure

The Terraform Module repository is a directory with a collection of Zip files, consisting of these main coordinates:

- namespace
- module
- Provider
- version.zip

Artifactory complies with the Terraform Module directory layout convention.

namespace/module-name/provider/version.zip

For example:

```
terraform-aws-modules/vpc/aws/2.0.0.zip
harshicorp/consul/aws/DEV-123.zip
```

#### 7.37.3.2 | Use Terraform Provider Registries

Providers are a set of plugins that interact with cloud providers, SaaS providers and other APIs.

##### Terraform Provider Repository Structure

The Terraform Provider repository is a directory with a collection of Zip files consisting of these main coordinates:

- Namespace
- Provider
- Version.zip
- Operating system
- Architecture

Artifactory complies with the Terraform Provider directory layout convention.

`namespace/provider/version/terraform-provider-{NAME}_{VERSION}_{OS}_{ARCH}.zip`

For example:

`/google/3.88.0/terraform-provider-google_3.88.0_darwin_amd64.zip`

### 7.37.3.3 | Set Up a Terraform Module/Provider Registry

You can set up the following repository types:

- Local Repository
- Remote Repository
- Virtual Repository

#### 7.37.3.3.1 | Set up a Local Terraform Module/Provider Registry

Local repositories enable you to deploy the Terraform Module or Provider as a zip archive. Artifactory calculates the metadata for all packages and indexes them to allow users to download these packages through the Terraform CLI.

1. To create a Terraform Module/Provider local repository, navigate to the **Administration** module.
2. Navigate to **Repositories** | **Repositories** | **Local** | **New Local Repository** and select **Terraform** as the **Package Type**.
3. In the Basic tab, from the **Terraform Registry Type** list:
  - Select **Module** to apply the `terraform-module-default` repository layout.
  - Select **Provider** to apply the `terraform-provider-default` repository layout.

#### 7.37.3.3.2 | Set Up a Remote Terraform Registry

Remote Repositories enable you to proxy and cache your remote Terraform Modules and Providers packages.

To specify that a Remote Repository supports Terraform packages, you will need to set its **Package Type** to **Terraform** when it is created.

Note that the Remote repository settings are identical for both Providers and Modules.

Follow these guidelines regarding the specific Terraform settings.

Field	Description
URL	The base URL of the Module storage API.  When using Smart remote repositories, set the URL to <base_Artifactory_URL>/repokey.
Git Providers	It is recommended to select Github when pointing to the official HashiCorp Terraform Registry.  Note that other Git Providers are not supported.
Registry URL	The base URL of the registry API.  When using Smart Remote Repositories, set the URL to <base_Artifactory_URL>/api/terraform/repokey  When using OpenTofu, set the URL to https://registry.opentofu.org/
Providers URL	The base URL of the Provider's storage API.  When using Smart remote repositories, set the URL to <base_Artifactory_URL>/api/terraform/repokey/providers  When using OpenTofu, set the URL to https://github.com/

For information on the common Remote repository fields, see [Remote Repositories](#).

#### Note

Starting from version 7.39.4, The default Terraform behavior is to override the source URL when initializing a module. It is not recommended to disable this feature, but if you need to, you can do so by setting this system property:

```
artifactory.terraform.override.modules.source.hostname=false
```

7.37.3.3.3 | [Set Up a Virtual Terraform Registry](#)

A **Virtual Repository** defined in Artifactory aggregates packages from both local and remote repositories.

This allows you to access both locally hosted Modules and Providers and remote proxied Terraform repositories from a single URL defined for the Virtual Repository.

To define a virtual Terraform repository, do the following:

1. Create a **Virtual Repositories**, and set the **Package Type** to be **Terraform**.
2. Select the underlying local and remote Terraform Module or Provider repositories to include in the **Basic** settings tab.

# JFrog Artifactory Documentation Displayed in the header

## 7.37.3.4 | Set Up Terraform Module/Provider Registry to Work With Artifactory

This section describes how to set up a Terraform module/provider registry to work with Artifactory. It includes the following guides:

- Configure Terraform Provider Registry with Artifactory Using OpenTofu
- Configure Terraform Provider Registry with Artifactory Using Terraform Client
- Deploy Terraform Resources
  - Deploy Terraform Providers
  - Deploy Terraform Modules
- Resolve Terraform Resources
  - Resolve Terraform Providers
  - Resolve Terraform Modules

### 7.37.3.4.1 | Configure Terraform Provider Registry with Artifactory Using OpenTofu

To configure your OpenTofu CLI to work against Artifactory:

1. Authenticate your client using the following command:

#### Note

Make sure to replace the placeholder with your JFrog Platform domain.

```
tofu login <YOUR_JFROG_DOMAIN>
```

2. To resolve the providers, add the following snippet to your Terraform configuration file, which can be found at `~/.terraformrc` (Linux and Unix) or `%APPDATA%/terraform.rc` (Windows):

#### Note

Make sure to replace the placeholders with your JFrog Platform domain and the Terraform repository you would like to use.

```
provider_installation {  
    direct {  
        exclude = ["registry.opentofu.org/*/*"]  
    }  
    network_mirror {  
        url = "<YOUR_JFROG_DOMAIN>/artifactory/api/terraform/<REPOSITORY_KEY>/providers/"  
    }  
}
```

### 7.37.3.4.2 | Configure Terraform Provider Registry with Artifactory Using Terraform Client

To configure your Terraform client to work against Artifactory:

1. Authenticate your client using the following command:

#### Note

Make sure to replace the placeholder with your JFrog Platform domain.

```
terraform login <YOUR_JFROG_DOMAIN>
```

2. To resolve the providers, add the following snippet to your Terraform configuration file, which can be found at `~/.terraformrc` (Linux and Unix) or `%APPDATA%/terraform.rc` (Windows):

#### Note

Make sure to replace the placeholders with your JFrog Platform domain and the Terraform repository you would like to use.

```
provider_installation {  
    direct {  
        exclude = ["registry.terraform.io/*/*"]  
    }  
    network_mirror {  
        url = "<YOUR_JFROG_DOMAIN>/artifactory/api/terraform/<REPOSITORY_KEY>/providers/"  
    }  
}
```

### 7.37.3.4.3 | Deploy Terraform Resources

You can deploy the following types of Terraform resources:

- Terraform modules
- Terraform providers

#### 7.37.3.4.3.1 | Deploy Terraform Providers

To deploy a Terraform provider to an Artifactory repository, use the following curl command:

#### Note

Make sure to replace the placeholders with your token, JFrog Platform domain, repository key, namespace, provider name, version, operating system, architecture, and path to the file on your machine.

# JFrog Artifactory Documentation Displayed in the header

```
curl -upm-admin:<TOKEN> -XPUT  
"https://<YOUR_JFROG_DOMAIN>/artifactory/<REPOSITORY_KEY>/<NAMESPACE>/<PROVIDER_NAME>/<VERSION>/terraform-provider-<PROVIDER_NAME>_<VERSION>_<OPERATING_SYSTEM>_<ARCHITECTURE>.zip" -T <PATH_TO_FILE>
```

For example:

```
curl -upm-admin:EKu8Bz5pCLLM8p1RhGqVeeDzrsHJpTCygccrrUCdFm00tkRqe4UCjU5s0Ke7T7km0 -X PUT  
"http://johnf.jfrog.io/artifactory/terraform-main-provider/hashicorp/null/3.1.0/terraform-provider-null_3.1.0_linux_amd64.zip" -T test-provider-linux-amd64.zip
```

7.37.3.4.3.2 | Deploy Terraform Modules

To deploy a Terraform module to an Artifactory repository, use the following curl command:

## Note

Make sure to replace the placeholders with your token, JFrog Platform domain, repository key, namespace, module name, provider name, version, and path to the file on your machine.

```
curl -upm-admin:<TOKEN> -XPUT  
"https://<YOUR_JFROG_DOMAIN>/artifactory/<REPOSITORY_KEY>/<NAMESPACE>/<MODULE_NAME>/<PROVIDER_NAME>/<VERSION>.zip" -T <PATH_TO_FILE>
```

For example:

```
curl -upm-admin:EKu8Bz5pCLLM8p1RhGqVeeDzrsHJpTCygccrrUCdFm00tkRqe4UCjU5s0Ke7T7km0 -X PUT  
"http://johnf.jfrog.io/artifactory/terraform-main-module-local-storage-rmmqrwdhfc/jfrog/test-module/test-provider/1.0.0.zip" -T test-module-1.0.0.zip
```

7.37.3.4.4 | Resolve Terraform Resources

You can resolve the following types of Terraform resources:

- Terraform modules
- Terraform providers

7.37.3.4.4.1 | Resolve Terraform Providers

To resolve a Terraform provider from an Artifactory repository, add the following information to your HCL file (`main.tf`):

## Note

Make sure to replace the placeholders with your namespace and provider name.

```
terraform {  
    required_providers {  
        <PROVIDER_NAME> = {  
            source = "<NAMESPACE>/<PROVIDER_NAME>"  
        }  
    }  
}
```

For example:

```
terraform {  
    required_providers {  
        docker = {  
            source = "kreuzwerker/docker"  
        }  
    }  
}
```

7.37.3.4.4.2 | Resolve Terraform Modules

To resolve a Terraform module from an Artifactory repository, add the following information to your HCL file (`main.tf`):

## Note

Make sure to replace the placeholders with your token, JFrog Platform domain, namespace, module name, and provider name.

```
module "module-name" {  
    source = "<YOUR_JFROG_DOMAIN>/<REPOSITORY>_<NAMESPACE>/<MODULE_NAME>/<PROVIDER_NAME>"  
}
```

For example:

```
module "vpc" {  
    source = "tflocal.com/terraform-remote-chlwfrvoj9_terraform-aws-modules/vpc/aws"  
}
```

7.37.3.5 | Generate an Access Token for Terraform

To use Artifactory with your Terraform CLI, you will need to generate an access token using the following two methods:

- Generating an Access Token Using Browser Login
- Manually Generating an Identity Token

# JFrog Artifactory Documentation

## Displayed in the header

The token will allow you to resolve Terraform module/provide packages using Terraform Registry, and to set up Artifactory as the remote state and locking provider using Terraform Backend repository.

Generating an Access Token Using Browser Login

1. Natively run the Terraform login to your Artifactory domain to generate an access token and save it in the Credentials file (`~/.terraform.d/credentials.tfrc.json`). When you run the login, the following page opens.

The Terraform command line login process is displayed.

2. You are routed to the JFrog Platform login page and the following page opens.

# JFrog Artifactory Documentation

## Displayed in the header

3. Click **Approve**. The following message is displayed.

4. Close the page and return to the command line to view the success message.

5. Next, perform a one-time edit to the `~/.terraformrc` file to ensure that the CLI searches for the providers only in the Artifactory. For this purpose, we recommend only pointing to the Terraform Virtual repository.

### Manually Generating an Identity Token

1. Generate an identity token that you can use to connect Artifactory to the Terraform CLI. For more details, please refer to User Profile - Identity Token.

2. Create a file named 'credentials.tfrc.json' in your Terraform directory, ('`~/.terraform.d/credentials.tfrc.json`' ).

3. Update the identity token you generated in the `credentials.tfrc.json` file, as in the example below:

```
#cat ~/.terraform.d/credentials.tfrc.json

{
  "credentials": {
    "ARTIFACTORY-DOMAIN": {
      "token": "IDENTITY-TOKEN"
    }
  }
}
```

### Note

If you have added the access token in `credentials.tfrc.json`, you do not need to login again using Terraform login `servername.jfrog.io`.

### 7.37.3.6 | Search for Terraform Packages

You can search for Terraform Module or Providers by their type, using the [Artifact Package Search](#).

### 7.37.3.7 | Calculate a Terraform Repository Metadata

You can trigger an asynchronous re-indexing of a local Terraform repository either through the UI or using the REST API.

This will also reindex the git index and, as a result, will also index the remote repositories.

In the [Artifact Tree Browser](#), select your Terraform repository, right-click and select **Recalculate Index** from the list. Requires Admin privileges.

To reindex a Terraform repository through the REST API refer to the following REST API: Calculate Terraform Index

# JFrog Artifactory Documentation

## Displayed in the header

### 7.37.3.8 | Publish Terraform Modules Using the JFrog CLI

JFrog CLI provides full support for packing Terraform modules and deploying them to JFrog Artifactory.

#### Required Software Versions

- JFrog CLI version 2.12.0 and above.
- JFrog Artifactory version 7.38.10 and above.

### 7.37.3.8.1 | Step 1: Set Up Terraform Repositories

Before you can use JFrog CLI to publish your Terraform modules to Artifactory, you first need to set the deployment repository for the project.

To set the repository:

1. CD to the root of the Terraform project.
2. Run the `jf terraform-config` command.

Argument/ Flag	Description
Command-name	<code>terraform-config</code>
Abbreviation	<code>tfc</code>
--global	[Default false] Set to true, if you'd like the configuration to be global (for all projects on the machine). Specific projects can override the global configuration.
--server-id-deploy	[Optional] Artifactory server ID for deployment. The server should be configured using the ' <code>jf c add</code> ' command.
--repo-deploy	[Optional] Repository for artifacts deployment.

For example, to set repositories for this Terraform project, run the following command:

```
jf tfc
```

To set repositories for all Terraform projects on this machine, run the following command:

```
jf tfc --global
```

### 7.37.3.8.2 | Step 2: Publish Terraform Modules to Artifactory

The `terraform publish` command packs and deploys the Terraform modules to the designated Terraform repository in Artifactory.

Before running the `terraform publish` command on a project for the first time, the project should be configured using the `terraform-config` command.

The modules are deployed to the configured Terraform repository according to the following layout.

Note that the `moduleName` is determined by the name of the module directory

```
namespace/moduleName/provider/tag.zip
```

To publish a specific module, access the Modules' root directory.

Before running `terraform publish`, cd into the directory which contains the Terraform modules. Note that those modules will later share common layout arguments (namespace, provider, and tag) in Artifactory.

#### Rules and Guidelines for Packing Terraform modules

- The File system is scanned from the current working directory.
- A directory that contains at least one file with a `.tf` extension is considered as a Terraform module and all its content will be packed in one zip file (including the submodules directories) and will be deployed to Artifactory.
- Recursive scanning does not run inside the module directory after it is packed, as a result, the submodules will not be packed and deployed separately to Artifactory.

#### Adding Exclusions Using Patterns

There is an option to set patterns to exclude modules that may contain:

- A directory that matches the pattern will not be scanned.
- Files that match the pattern won't be packed in the relevant zip file.

The following table lists the command arguments and flags:

# JFrog Artifactory Documentation

## Displayed in the header

Argument/ Flag	Description
Command-name	terraform publish
Abbreviation	tfp
--exclusions	[Optional] A list of Semicolon-separated exclude patterns. Allows using wildcards.
Command arguments	
--namespace	The namespace of the Terraform project that its modules are being published.
--provider	The provider of the Terraform project that it's modules are being published.
--tag	The tag of the modules that are being published.

### Example of Packing and Publishing Terraform Modules

Before running this command on a project for the first time, the project should be configured using the `terraform-config` command.

To pack and publish the Terraform module, run the following command.

This command packs and deploys all the modules under the current working directory.

The modules will be published to the configured repository, using this layout: `tera/moduleName/aws/v0.1.2.zip`.

Modules and files which include `test` and `ignore` will be excluded (read more in the "exclusions" section).

```
jf tf p -namespace=tera -provider=aws -tag=v0.1.2 -exclusions="*test*;*ignore*"
```

For more information try this [Terraform project example](#).

#### 7.37.4 | Terraform Backend Repository

From JFrog Artifactory 7.38.4, the Terraform Backend repository serves as a dedicated Remote State Storage Provider. It works together with and in parallel to the Terraform registry, which also serves as the dedicated Terraform registry for hosting your modules and providers in the JFrog Platform. For more information, see [Terraform Backends](#).

Each Terraform Module can have an associated Backend that defines how operations are executed and a state file that tracks the resources created by your configuration and maps them to real-world resources. Certain backends support multiple named workspaces, allowing multiple states to be associated with a single configuration. The configuration still has only one backend, but multiple distinct instances of that configuration can be deployed without configuring a new backend or changing authentication credentials.

##### Did you know?

Each Terraform configuration can specify a backend, which defines where and how operations are performed including where the snapshots are stored and more. Terraform uses persistent State data to keep track of the resources it manages and includes information on how real-world infrastructure objects correspond to the resources in a configuration. All users working on the collection of infrastructure resources need access to the same state data. For more information, see [Terraform States](#).

##### State Locking

Terraform automatically locks all your operations that have the capability to change the State to prevent others from acquiring the lock and potentially damaging your state. To learn how Artifactory supports State Locking, see [Viewing State and Lock Information and History](#)

#### JFrog Terraform Backend Repository Meets the Hashicorp Standards

In the first half of 2022, Hashicorp announced the deprecation of a number of legacy providers, including the legacy `artifactory` provider, which served as a basic backend that only stored States in a generic JFrog repository, created and maintained by Hashicorp.

To avoid confusion, please note that the official JFrog Artifactory Terraform Backend repository, described in this article, is unaffected by the Hashicorp `artifactory` backend provider deprecation action.

As part of JFrog's alignment with the common Hashicorp main practices, the JFrog Terraform Backend repository supports the official Hashicorp enterprise-grade Terraform backend provider, and supports features such as locking, encoding of the data, smart comparisons, and additional common practices.

#### Federated Repositories Not Supported for Artifactory Backend Repositories

It is not possible to connect a Terraform Backend repository to a Federated repository. This limitation prevents inconsistencies in the system state, which could lead to unexpected behavior or errors.

##### Note

Replication of Terraform Backend Repositories is fully supported from Artifactory version 7.77.x and later.

#### 7.37.4.1 | Terraform Backend Repository Structure

The Terraform Backend repository is a directory with a collection of workspaces consisting of these main coordinates:

# JFrog Artifactory Documentation

## Displayed in the header

- backend: Set as Remote by definition
- hostname: Your Artifactory domain name
- organization: Backend Artifactory Repository name
- prefix: Allows users to add a user-defined prefix when working with multiple workspace under the same prefix.

```
terraform {
  backend "remote" {
    hostname = "my_artifactory_domain.org"
    organization = "backend repository name"
    workspaces {
      prefix = "my-prefix-"
    }
  }
}
```

### 7.37.4.2 | Set Up a Local Terraform Backend Repository

Local repositories enable you to deploy Terraform Backend Zip files. Artifactory calculates the metadata for all file and indexes them to allow users to download these packages through the Terraform CLI. Remote and virtual repositories are not supported for Terraform Backend repositories.

1. To create a Terraform Backend local repository, navigate to the **Administration** module.
2. Navigate to **Repositories|Repositories|Local|New Local Repository** and select **Terraform BE** as the **Package Type**.

### 7.37.4.3 | Set Up Terraform Backend Repository to Work With Artifactory

This section describes how to set up a Terraform Backend Registry to work with Artifactory. It includes the following guides:

- [Configure Terraform Backend Repository with Artifactory Using OpenTofu](#)
- [Configure Terraform Backend Repository with Artifactory Using Terraform Client](#)

For more information about deploying and resolving Terraform resources, see:

- [Deploy Terraform Resources](#)
- [Resolve Terraform Resources](#)

#### 7.37.4.3.1 | Configure Terraform Backend Repository with Artifactory Using OpenTofu

To configure your OpenTofu CLI to work against Artifactory as a remote state storage and locking provider:

1. Authenticate your client using the following command:

#### Note

Make sure to replace the placeholder with your JFrog Platform domain.

```
tofu login <YOUR_JFROG_DOMAIN>
```

# JFrog Artifactory Documentation

## Displayed in the header

2. To configure the Terraform CLI to work with Artifactory, add the following information to your HCL file (`main.tf`):

### Note

Make sure to replace the placeholders with your JFrog Platform domain, the Terraform repository, and the prefix you would like to use.

```
terraform {  
    backend "remote" {  
        hostname = "<YOUR_JFROG_DOMAIN>"  
        organization = "<REPOSITORY_KEY>"  
        workspaces {  
            prefix = "<PREFIX>"  
        }  
    }  
}
```

For example:

```
terraform {  
    backend "remote" {  
        hostname = "johnf.jfrog.io"  
        organization = "tb-local"  
        workspaces {  
            prefix = "jfrog-"  
        }  
    }  
}
```

### 7.37.4.3.2 | Configure Terraform Backend Repository with Artifactory Using Terraform Client

To configure your Terraform client to work against Artifactory as a remote state storage and locking provider:

1. Authenticate your client using the following command:

### Note

Make sure to replace the placeholder with your JFrog Platform domain.

```
terraform login <YOUR_JFROG_DOMAIN>
```

2. To configure the Terraform CLI to work with Artifactory, add the following information to your HCL file (`main.tf`):

### Note

Make sure to replace the placeholders with your JFrog Platform domain, the Terraform repository, and the prefix you would like to use.

```
terraform {  
    backend "remote" {  
        hostname = "<YOUR_JFROG_DOMAIN>"  
        organization = "<REPOSITORY_KEY>"  
        workspaces {  
            prefix = "<PREFIX>"  
        }  
    }  
}
```

For example:

```
terraform {  
    backend "remote" {  
        hostname = "johnf.jfrog.io"  
        organization = "tb-local"  
        workspaces {  
            prefix = "jfrog-"  
        }  
    }  
}
```

### 7.37.4.4 | Generate an Access Token for Terraform

To use Artifactory with your Terraform CLI, you will need to generate an access token using the following two methods:

- [Generating an Access Token Using Browser Login](#)
- [Manually Generating an Identity Token](#)

The token will allow you to resolve Terraform module/provide packages using Terraform Registry, and to set up Artifactory as the remote state and locking provider using Terraform Backend repository.

[Generating an Access Token Using Browser Login](#)

1. Natively run the Terraform login to your Artifactory domain to generate an access token and save it in the Credentials file (`~/.terraform.d/credentials.tfrc.json`). When you run the login, the following page opens.

The Terraform command line login process is displayed.

2. You are routed to the JFrog Platform login page and the following page opens.

3. Click **Approve**. The following message is displayed.

4. Close the page and return to the command line to view the success message.

5. Next, perform a one-time edit to the `~/.terraformrc` file to ensure that the CLI searches for the providers only in the Artifactory. For this purpose, we recommend only pointing to the Terraform Virtual repository.

#### Manually Generating an Identity Token

1. Generate an identity token that you can use to connect Artifactory to the Terraform CLI. For more details, please refer to User Profile - Identity Token.
2. Create a file named 'credentials.tfrc.json' in your Terraform directory, ('`~/.terraform.d/credentials.tfrc.json`' ).
3. Update the identity token you generated in the `credentials.tfrc.json` file, as in the example below:

```
#cat ~/.terraform.d/credentials.tfrc.json

{
  "credentials": {
    "ARTIFACTORY-DOMAIN": {
      "token": "IDENTITY-TOKEN"
    }
  }
}
```

#### Note

If you have added the access token in `credentials.tfrc.json`, you do not need to login again using Terraform login `servername.jfrog.io`.

#### 7.37.4.5 | View Individual Terraform Workspace Information

Artifactory lets you view selected metadata of a Terraform Backend workspace directly from the UI.

In the Artifact Repository Browser, select your local Terraform Backend repository and scroll down to find and select the package you want to inspect.

The metadata is displayed in the **Terraform Info** tab, or viewed in the **Packages** view.

#### View State and Lock Information and History

Artifactory automatically generates a `state.json` file when a `workspace.json` is deployed. The latest file is declared as the `state.latest.json` and all previous states are renamed with a timestamp indicating the time they were created, for example, `state.1640018380463.json`.

Artifactory provides information about the workspace deployment process.

In Artifactory, while performing Terraform configuration changes you will notice that the `locked_by` property together with the name of the user performing the change is displayed in the Properties tab of the Workspace.

## 7.38 | Vagrant Repositories

In addition to 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.

### 7.38.1 | Set Up a Vagrant Repository

You can set up local Vagrant repositories. Remote and Virtual repositories are not supported for Vagrant.

#### 7.38.1.1 | Set Up Local Vagrant Repositories

To create a local Vagrant repository to host your Vagrant boxes, from the **Administration** module go to **Repositories | Repositories | Local** and create a new Local Repository and set **Vagrant** as the **Package Type**.

### 7.38.2 | Deploy Vagrant Boxes

You can deploy Vagrant boxes using the following methods:

- Using the JFrog Platform WebUI
- Using Matrix Parameters

#### 7.38.2.1 | Deploy a Vagrant Package Using the JFrog Platform WebUI

To deploy a Vagrant box to Artifactory, in the **Application** module, under **Artifactory | Artifacts** 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**.

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
```

Vagrant 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.

#### 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:



#### 7.38.2.2 | Deploy a Vagrant 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={provider};box_version={version}"
```

For example, to upload the box **precise64-virtualbox-1.0.0.box**, and specify that it's 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 for Vagrant

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_provider=[provider];box_version=[version]"
```

Parameter	Description
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 provider (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)

### 7.38.3 | Provision 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 its 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](#)

### 7.38.4 | Configure Authenticated Access to Vagrant 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. This token is a Base64 encoded string of the user credentials (formatted `username:password`).

b. From [Artifactory 7.25.7](#), you will need to create an Access Token via the JFrog Platform UI and use that token to authorize Vagrant access.

#### 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=YWRtaW46QVAzWGHzWmlDU29NVmtaQ2dCZEY3XXXXXXX
export VAGRANT_SERVER_URL=http://localhost:8081/api/vagrant/vagrant-local
```

#### Both environment variables are required

When using Artifactory with authenticated access (i.e. anonymous access is disabled), both of these environment variables are required. If either of them is not set, the Vagrant client will not be able to access Artifactory and a 401 error message will be generated.

## 7.39 | VCS Repositories

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 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.

### 7.39.1 | Set Up a VCS Repository

You can set up remote VCS repositories. Artifactory does not support local or virtual repositories for VCS.

This topic contains the following guides:

- [Configure VCS Repository Layout](#)
- [Set Up Remote VCS Repositories](#)
- [Use VCS To Proxy Git Providers](#)

#### 7.39.1.1 | Configure VCS Repository Layout

VCS repositories require an appropriate repository layout to support a more hierarchical layout of the cached items.

# JFrog Artifactory Documentation

## Displayed in the header

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.

**Integration Benefits** JFrog Artifactory and Git

**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**:

You can configure the Custom Layout, or simply copy the below code snippet into the relevant section in the **Administration** module, under **Repository Management | Layouts**.

```
<repoLayout>
    <name>vcs-default</name>
    <artifactPathPattern>[orgPath]/[module]/[refs<tags|branches>]/[baseRev]/[module]-[baseRev](-[fileIntegRev])(-[classifier]).[ext]</artifactPathPattern>
    <distinctiveDescriptorPathPattern>false</distinctiveDescriptorPathPattern>
    <folderIntegrationRevisionRegExp>.*</folderIntegrationRevisionRegExp>
    <fileIntegrationRevisionRegExp>[a-zA-Z0-9]{40}</fileIntegrationRevisionRegExp>
</repoLayout>
```

If a repository package layout is in a corresponding folder hierarchy, the Artifactory Version Cleanup tool correctly detects previously installed versions.

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
} ]
}
```

### 7.39.1.2 | Set Up Remote VCS Repositories

You need to create a **Remote Repository** which serves as a caching proxy for [github.com](https://github.com). If necessary, you can do the same for [bitbucket.org](https://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](https://github.com) or an self-hosted GitHub Enterprise repository, follow the steps below:

1. In the **Administration** module, under **Repositories | Repositories | Remote**, click **New Remote Repository** 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:



3. Under VCS Settings, select the GitHub provider in the **Git Provider** field and click **Save & Finish**.

### 7.39.1.3 | Use VCS To Proxy 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:



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>.tar.gz>

Therefore, the custom download path configured for Artifactory should be {0}/{1}/archive/{2}.{3}

### 7.39.2 | REST API Support for VCS

#### 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 Cloud, the URL would be:

`https://<server name>.jfrog.io/artifactory/api/vcs/<repository key>`

Artifactory exposes REST APIs that let you do the following with VCS repositories:

- List all tags
- List all branches
- Download References file
- Download a specific tag
- Download a file within a tag
- Download a specific branch
- Download a file within a branch
- Download a release
- Download a commit

To help you build the API call correctly, you can select the VCS repository you want to interact with and click **Set Me Up**.

## 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/lsystems/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"
```

## 7.39.3 | Access 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.

## 8 | Artifact Management

JFrog Artifactory serves as a single source of truth for build artifacts across your organization, and deploys and resolves artifacts by configuring your package manager and CI tools to work opposite Artifactory.

Using Artifactory to store and manage your artifacts provides the following benefits:

- Universal support with over 30 integrated package types.
- Complete metadata on every package allows for wide observability and traceability.
- Control over the components your developers can access and leverage in your software development lifecycle.
- Unlimited scalability with a variety of enterprise-scale storage capabilities.
- Security enhancement through a native integration with JFrog Xray.

# JFrog Artifactory Documentation

## Displayed in the header

For more benefits and information, see [JFrog Artifactory](#).

### 8.1 | Storing and Using Your Artifacts

Artifactory stores your first-party artifacts in [local repositories](#), which are locally-managed repositories into which you can deploy your artifacts. After configuring a local repository, you can deploy and resolve your artifacts directly from the JFrog Platform UI or API.

This section reviews how to manage artifacts using the following options:

- [Use a Package Manager Client](#)
- [Browse and Search Artifacts](#)
- [Use the REST API for Managing Artifacts](#)

#### Use a Package Manager Client

You create a connection between Artifactory and the package manager client you are working with by configuring Artifactory as a local repository for the package manager client. Once you configure a local repository, you can deploy and resolve artifacts directly from Artifactory.

To configure a client, go to the [Artifacts](#) page, click [Set Me Up](#), and follow the instructions that appear. Each package manager client has slightly different instructions, which Artifactory will generate specifically for your instance.

In most cases, the configuration process has the following structure:

- Configure
- Deploy
- Resolve

#### Configure

First, set the appropriate Artifactory repository as the default registry for the package manager client you are working with.

Next, enter your Artifactory and package credentials in the package manager client and establish a link between the two.

#### Deploy

To deploy your packages to the Artifactory local repository, use an API or CLI command to deploy the package type you selected.

#### Resolve

To resolve your packages from the Artifactory local repository, use a command to resolve the package type you selected.

After adding an artifact to a local repository, you will be able to access it using an Artifactory URL. Refer to the Set Me Up section for the package type you are working with for specific instructions, URLs, and file paths.

#### Example: Docker Local Repository

To store and use your Docker images on Artifactory, configure Artifactory as a Docker repository, as follows:

1. To create a new Docker local repository, access the JFrog UI, navigate to [Administration | Repositories | Add Repositories](#), and select [Local Repository](#).
2. Fill in the repository key and URL, and any other repository settings you would like to apply, and click [Create Local Repository](#).
3. Navigate to [Application | Artifactory | Artifacts | Set Me Up](#), and select [Docker](#) from the packages list.
4. Select your repository from the drop-down menu. Instructions for configuring the client will appear under the [Configure](#) tab. Follow the instructions to establish the connection between your Docker client and Artifactory.
5. Once you establish the connection between your Docker client and Artifactory, you can tag, push, and pull your Docker images using Artifactory.

#### Browse and Search Artifacts

After adding artifacts to a local repository, you can browse them using the JFrog Platform UI. For more information, see [Browsing Artifacts](#).

You can also search for specific remote cached resources according to the package metadata, searching by version, permissions, type, and more. For more information, see [Application Search](#).

#### Artifactory Query Language

Artifactory Query Language (AQL) is a standardized language that allows you to extract any data related to your artifacts and builds stored in Artifactory. For more information, see [Artifactory Query Language](#).

#### Use the REST API for Managing Artifacts

Artifactory allows you to use REST API endpoints to perform a vast number of actions on a JFrog instance. This helps enable automation across your pipelines. For more information, see [Artifactory REST API](#).

### 8.2 | Saving and Using Third-Party Dependencies

Nearly all code references third-party dependencies. With JFrog, you can proxy packages from remote resources in your local cache for easier use, by configuring your package manager client to work opposite JFrog Artifactory.

Proxying third-party dependencies provides the following benefits:

# JFrog Artifactory Documentation

## Displayed in the header

- Local caching allows for a faster and more stable build process.
- High availability for cached dependencies, even if the remote resource is down or unavailable.
- Checksum-based storage allows for lower file system resource consumption.
- Rich metadata and properties allow for easy and accurate search.

Artifactory stores third-party dependencies in [remote repositories](#), which are repositories that serve as caching proxies for packages taken from remote resources such as public registries (ie. Maven Central, crates.io). After configuring a remote repository, you can browse and resolve third-party dependencies directly from the JFrog Platform UI or API.

### 8.2.1 | Configure a Package Manager Client

To configure a client to work with Artifactory, go to the [Artifacts](#) page, click **Set Me Up**, and follow the instructions that appear. Each package manager client has slightly different instructions, which Artifactory will generate specifically for your instance.

In most cases, the configuration process has the following structure:

- Configure
- Resolve

#### Configure

First, set the appropriate Artifactory repository as the default registry for the package manager client you are working with.

Next, enter your Artifactory and package credentials to the package manager client, and establish a link between the two.

#### Resolve

You can cache and use third-party packages using an Artifactory dedicated URL. For the package type you are working with for the specific instructions, URLs, and file paths, refer to the [SetMeUp](#) section.

#### Example: npm Remote Repository

To proxy [npm packages](#) using Artifactory, you will need to configure Artifactory as an npm remote repository:

1. To create a new npm remote repository, navigate to [Administration | Repositories | Add Repositories](#), and select **Remote Repository**.
2. Fill in the repository key and URL, and any other repository settings you would like to apply, and click **Create Remote Repository**.
3. Navigate to [Application | Artifactory | Artifacts | Set Me Up](#), and select **npm** from the packages list.
4. Select your repository from the drop-down menu. Instructions for configuring the client will appear under the **Configure** tab. Follow the instructions to establish the connection between your npm client and Artifactory.
5. Once you establish the connection between your npm client and Artifactory, you can use the command `npm install [PACKAGE_NAME]`; to proxy packages from npm Registry or any other resource in your Artifactory repository.

### 8.2.2 | Browse Third-Party Dependencies

#### Browse Artifacts

After adding artifacts to a remote repository, you can browse them using the JFrog Platform UI. For more information, see [Browsing Artifacts](#).

#### Note

The remote resource that Artifactory proxies may not support remote browsing. For more information, see [Remote Browsing](#).

#### Search Packages

You can search for specific remote cached resources according to the package metadata, searching by version, permissions, type, and more. For more information, see [Application Search](#).

### 8.2.3 | Best Practices for Managing Third-Party Dependencies

#### Smart Remote Repositories

To access your remote resources in other instances of Artifactory, you can create a [smart remote repository](#), which is a repository that proxies a repository from another Artifactory instance. smart remote repositories allow you to synchronize the properties and see download statistics.

#### Virtual Repositories

To aggregate resources from local and remote repositories in one place for a single point of resolution, you can create a [virtual repository](#). For more information, see [Onboarding Best Practices: JFrog Artifactory](#).

## 8.3 | Working with JFrog Properties

Properties are user-customizable fields that can be any string and can have any value(s). Properties are used extensively in the JFrog REST APIs and CLI for JFrog Artifactory (command line tool), the Artifactory Query Language (AQL), and in the JFrog Platform UI.

For Artifactory, you can place properties on both artifacts and folders; setting and deleting properties is supported by local repositories or local-cache repositories. In addition, while you cannot set or delete properties on virtual repositories, you can retrieve them. You can assign properties using the [UI](#), via [REST API](#), via the [CLI](#), or [on deployment](#) using Matrix Parameters. Properties can also be used to control Artifacts resolution.

# JFrog Artifactory Documentation

## Displayed in the header

In addition, properties are searchable and can be combined with Smart searches to search for items based on their properties and then to manipulate all the items in the search result in one go. Properties can also be used with integrations/extensions, such as user plugins, Jenkins plugin, etc.

### 8.3.1 | Properties Use Cases

JFrog properties can be used in many different implementations that fall under two main use cases:

- Explicit properties
- Implicit properties

#### Explicit Properties

Explicit properties are set by the customer, are customizable, and provide functions that are Searchable and Actionable. These properties can be connected, for example, to artifacts, and then allow for promotion, state management, additional organization information, and more.

- **Locating items in JFrog using properties:** By assigning a property to an item, you can easily search for the item in JFrog using that property.
- **Combining several packages:** Assign the same property to several packages, and then group the packages into one under the assigned property.
- **Applying values to multiple locations without hardcoding the values:** Properties provide the flexibility of identifying an item without hardcoding its values.
- **Creating and applying properties as needed:** Create and apply properties to any stage or build.

#### Implicit Properties

Implicit properties are JFrog Platform-generated properties and are predefined in the system, such as Last Update, Last Upload (e.g., cleanup based on these metadata). For a complete list of implicit properties and their attributes, see GraphQL.

### 8.3.2 | View and Search for Properties in the UI

There are a number of ways to search and view properties in the JFrog Platform UI.

#### Permissions

Only users who have *Annotate Permissions* can add and edit properties in JFrog; for more information, see Permissions.

#### View Properties in the Artifact Tree Browsers

When selecting any item in the tree browser, click its **Properties** tab to view or edit the properties attached to the item (if any have been set).

The properties appear in the Properties list at the bottom of the window..

#### Search for Properties in the UI

1. To search for properties in the UI, go to **Artifactory** in the Application tab and select **Artifacts**.
2. Click the **Search** filter function, and then in the Type dropdown, select **Property**.

This opens the Search Artifacts dialog.

3. From the Repository dropdown, select one or more repositories.
4. Optional:
  - a. To search for a specific property, click **+Add** to add the key and value of the property.
  - b. To search for a specific property-set, click **+Add** to add the key and value of the property set.
5. Click **Search** to begin searching.

#### 8.3.3 | Add and Delete Properties

##### Note

Property keys are limited up to 255 characters and property values are limited up to 2,400 characters. Using properties with values over this limit might cause backend issues.

##### Add Properties via the UI

When selecting any item in the tree browser, you can view its **Properties** tab to add and delete the properties attached to the item.

1. To add a property, open any item in the tree browser and go to the item's **Properties** tab.
2. In the **Add Property** field, enter a name and value for the property.

You can also add multi-value properties, by entering the values separated with a semi-colon ( ; ).
3. **Optional:** To add the property to the selected folder as well as to all of the artifacts, folders, and sub-folders under this folder, select the **Recursive** option.
4. Click **Add** to add the new property.

##### Note

You can add multiple properties to a single item; all properties will then appear at the bottom of the item Properties tab.

##### Delete a Property or Remove Recursively

- To delete a property, go to the list of properties, select the property, and then click **Delete** or click the **x** button to the right of the property.

You will be asked to confirm your deletion.
- To remove recursively, click **Delete Recursively** or the Delete Recursively icon to the right of the property.

You will be asked to confirm your deletion.

##### Add Properties to Different Types of Resources

You can add properties to different resources including packages, builds, Release Bundles v1, and artifacts (for example, to binary files). Adding properties to Release Bundles v1 is a particularly powerful feature since you can use the Recursive option to apply the property to all of the artifacts, folders, and sub-folders under the folder.

## Note

Properties cannot be added to Release Bundles v2, which are immutable. For information about the differences between Release Bundles v1 and v2, see [Types of Release Bundles](#).

### 8.3.4 | Use Properties in JFrog

This section provides some examples of how to use properties in JFrog through a wide variety of interfaces and options.

#### Use Properties in the REST APIs

Properties are a special form of metadata that are stored on items just like any metadata - but in XML form. 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 uses the properties root tag and has a very simple format. As such, you can set, retrieve and remove properties from repository items via REST API, as you would do with any other XML-based metadata. For more information, see [Artifactory REST APIs](#).

#### Use Properties in the CLI

JFrog CLI is a compact and smart client, which provides a simple interface for automating access to JFrog products. The `CLI` command enables you to set properties on any artifact in Artifactory. For more information, see [Setting Properties on Files](#).

#### Use Properties in the AQL

An AQL resource creates an Artifactory query using the Artifactory Query Language (AQL). This type of resource can be used in the configuration of a `FileSpec` resource to specify file properties to match against, and as an `inputresource` for the `CreateReleaseBundle` step to define the query that is used to create a release bundle. For more information, see [Aql](#).

#### Use Properties in the User Plugins

User plugins written in Groovy enable you to adapt Artifactory's behavior to meet your own needs, including responding to any storage events on items and properties. For more information, see [User Plugins](#).

#### Use 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 UI you see when setting a property value and when searching for property values; as such, using searchable properties in artifact management is a very powerful feature. For more information, see [Property Sets](#).

#### Matrix Parameters

Matrix parameters are key-value pairs separated by a semicolon (;) that you can place anywhere on a URL. For more information, see [Using Properties in Deployment and Resolution](#).

### 8.3.5 | Property Sets

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.

#### Create a Property Set

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.

#### Note

**Properties are for Guiding, not for Restricting**

# JFrog Artifactory Documentation

## Displayed in the header

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.

To create a new property set, go to **Admin | Artifactory | General | Property Sets** and click **New**.

Add a Property Set Name, click **New** and add properties. The newly created property set will be available to use and add to a repository.

### Assign Property Sets to a Repository

To assign a property set to a repository which can then be selected from with the Artifact browser, go to **Administration module | Repositories** and select a repository.

In the **Advanced tab**, select the property sets you would like to be available under this repository.

#### Add Property Sets to a Specific Repository

To add a property set to a specific repository, go to **Application** module | **Artifacts** and select a repository.

In the **Properties tab**, select the property sets you would like to add.

#### Attach 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.

i

You can edit the value of any property and click **Save**.

#### Attach and Read 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, update, and remove properties from repository items via REST API, as you would do with any other XML-based metadata.

#### 8.3.6 | Using Properties in Deployment and Resolution

##### Introducing Matrix Parameters

Matrix parameters are key-value pair parameters separated by a semicolon (;) that you can place anywhere in 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 a part of deployment
2. Controlling artifact resolution using matrix parameters

##### Dynamically Add 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).

# JFrog Artifactory Documentation

## Displayed in the header

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
color=black	color=black	OK (200)
None or height=50	color=black	OK (200)
color=red	color=black	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)

## 8.4 | Browsing Artifacts

The [Artifact Repository Browser](#) page provides two ways to browse through repositories:

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

Both browsing modes 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](#).

### Disabling the Native Browser UI

You can disable the native browser UI feature in `artifactory.system.properties` by setting the following system property to `false`:`artifactory.redirect.native.browser.requests.to.ui=false`.

Both the Tree and Simple browsing modes 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

#### 8.4.1 | How to Apply Filters

Filters enable you to reduce the number of items displayed in the artifact tree based on your specified criteria.

##### To filter by name:

1. In the Application module, select **Artifactory > Artifacts**.
2. Enter the name by which to filter the tree in the field provided.

##### To filter and sort by type:

1. In the Application module, select **Artifactory > Artifacts**.

2. Click the filter icon  to open the filter window.

3. Select one or more filter criteria from the package and repository lists.

4. Select a sorting option:

- Repository Type
- Repository Key
- Package Type

**Note**

Filters are saved as cookies in your browser.

#### 8.4.2 | How to Use Favorite Repositories

You can view only the repositories you need by customizing the Artifact Repository Browser with your favorite repositories, and applying sort and filter options. Use as many different favorite, sort and filter combinations to narrow down the Artifact tree to display exactly what you need. Note that your filters and favorites are saved as cookies in your browser cache.

To tag a repository as a favorite:

Select the repository in the left pane, right-click and select **Add to Favorites**.

To view the list of favorite repositories, click the **Favorites** icon.

## JFrog Artifactory Documentation Displayed in the header

### 8.4.3 | How to Use Tree Browsing

Tree browsing lets you drill down the repository hierarchy and display full information about each level. 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 chosen item. For example, if you select an npm package, an npm Info tab displays information specific to npm packages, similar to NuGet, RubyGems, and other supported package formats.

#### Collapse All

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

#### Sort the Tree Hierarchy

The default order in which repositories appear in the Tree hierarchy is: virtual, local, remote, and cached.

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=cached,remote,local,virtual
```

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

### 8.4.4 | Simple Browsing

Simple browsing 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.

### 8.4.5 | 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 Artifact Repository Browser (indicated in red in the screenshot below).

#### Artifact Repository Browser

## List Browser

### 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`

### 8.4.6 | Remote Browsing

For a Smart Remote Repository, Artifactory lets you navigate the contents of the repository on the remote Artifactory instance 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

### Initial responsiveness of remote repositories

Initial remote browsing might 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.

#### 8.4.7 | View Artifact Information

Displays the Artifact metadata. Regardless of whether you select the item in Tree browsing mode or Simple browsing mode, one or more of the following tabs appear containing the metadata associated with the selected item.

#### General Tab

Field	Description
Info	General Information including download statistics such as the total number of downloads, timestamp of last download and the last user who downloaded.  <b>Important</b>  In certain cases (particularly when working with large artifacts), the Created timestamp might be <i>later</i> than the Last Modified timestamp. This can occur because the Last Modified timestamp records when the upload began, whereas the Created timestamp is set only when the upload is complete and committed to the database.
Dependency Declaration	For Maven artifacts, this section provides code snippets for common build tools' dependency declaration.
Virtual Repository Associations	Indicates which virtual repositories "contain" the selected artifact.
Checksums	Displays SHA1, SHA-256 and MD5 checksums automatically.

#### Effective Permissions Tab

Displays the list of permissions in the context of users, groups or permission targets on the artifact level. For more information, see Permissions.

#### Xray Tab

Displays the status of the most recent scan, a summary of any security issues encountered, and the contents of the artifact's SBOM (software bill of materials). For more information, see [Xray Scan Results](#).

#### Evidence Tab

Displays a list of evidence files associated with the selected item. For more information, see [View the Artifact Evidence Table](#).

### Properties Tab

Displays the list of properties annotating the selected item.

### 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. For more information, see [Property Sets](#).

### Followers Tab

Displays the list of users following this item.

The Followers 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 Followers from the 'Followers' tab in the tree browser. Followers or folders intercept changes on all children. An admin or users with the "Manage" permission can view and manage followers via the 'Followers' tab in the tree browser.

Follow 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 followed item(s).

### Builds Tab

Displays the list of builds that either produce or use the selected item.

### 8.4.8 | View Xray Data on Artifacts

You can view Xray data on the artifact level.

Indicates if, and when the last time the selected artifact was scanned by Xray, as well as the **Top Severity** for any vulnerabilities detected.

To override download blocking for a specific artifact, click **Ignore Violation** located on the right far side of the selected violation. For more information, see [Analyzing Resource Scan Results](#).

### View a List of Artifacts Containing Vulnerabilities

Use the Security & Compliance search to view a list of artifacts that contain Xray vulnerabilities. For more information, see [Searching for Scanned Resources](#).

### 8.4.9 | Restoring Artifacts from the Trash Can

You can easily recover items that have inadvertently been deleted from local repositories from the trash can.

The trash can can be enabled or disabled in the [Trash Can Settings](#) by an administrator or a user with the **Manage** permission.

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-clicking on an item in the trash can gives you the option to refresh, restore it to its original location, or permanently delete it.

Right-clicking 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 that it remains visible even if you scroll the tree.

### 8.4.10 | WebDAV Browsing

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

## 8.5 | Manipulating Artifacts

The Artifact Repository Browser provides a set of actions including moving, copying, and deletion of artifacts to keep your repositories consistent and coherent. When an artifact is moved, copied or deleted, the corresponding metadata descriptors (such as *maven-metadata.xml*, RubyGems, Npm, and more) are immediately and automatically updated 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.

#### 8.5.1 | Download 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.

Field	Description
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://&lt;ARTIFACTORY&gt;/artifactory/&lt;REPOSITORY_KEY&gt;">http://&lt;ARTIFACTORY&gt;/artifactory/&lt;REPOSITORY_KEY&gt;</a>), depending on one of the below pre-conditions:</p> <ol style="list-style-type: none"><li>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).</li><li>2.The repository <b>Checksum Policy</b> is set to "Trust Server Generated Checksums".</li></ol> <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.

# JFrog Artifactory Documentation

## Displayed in the header

### Configure Folder Download

An Artifactory administrator can enable complete folder download in the **Administration** tab under **Artifactory | General | Settings**. This configuration will apply to all Artifactory users.

Field	Description
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.

### Block Folder Download

When trying to download a folder, if any of the artifacts in that folder are blocked for download, then downloading the folder will fail with an HTTP FORBIDDEN (403) error.

#### 8.5.2 | Move and Copy Artifacts

To move or copy an artifact or folder, select it in the Tree Browser and then click **Move Content** or **Copy Content** 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 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.

#### Simulate a Move or Copy

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**.

# JFrog Artifactory Documentation

## Displayed in the header

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

### Permissions required

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

#### 8.5.3 | Delete a Single Item

To delete a single artifact or directory, select the item in the Tree Browser, and click **Delete Content** 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.

#### 8.5.4 | Delete 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. Deleting 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 metadata 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 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 1,000 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.

## 8.6 | Deploying Artifacts

In the Artifact Repository Browser, you can deploy artifacts into a local repository from the **Artifacts** module by clicking **Deploy** to display the **Deploy** dialog. Artifacts can be deployed individually or in multiples.

#### Important

JFrog does not support deploying artifacts larger than 100 MB from the Artifactory User Interface. For artifacts larger than 100 MB, use either the JFrog CLI or the native package manager.

Regarding artifact names:

- Avoid the following special characters, which can lead to potential errors: | \* ? "
- & can be used as part of the name but not as a path token.
- A space before / is not allowed.

#### Using Import to "deploy" an entire repository

If you want to "deploy" an entire repository, import it using the Import Repository feature in the **Administration** module under **Artifactory | Import & Export | Repositories**.

You can also deploy artifacts to any repository using the Artifactory REST API. See this example for a quick start.

#### Uploading Non-Conformant Content

Repositories of each package type have built-in logic for parsing metadata, creating index files, and optimizing performance for packages of that specific type. Uploading non-conformant content, such as images, text files, and other resources that are not wrapped in the proper package format can impact indexing and reduce performance. Therefore, it is strongly recommended to upload generic content to a Generic repository.

### 8.6.1 | Deploy a Single Artifact

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

#### Deploy 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.

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.

#### Deploy 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**.

# JFrog Artifactory Documentation

## Displayed in the header

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 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
```

### Deploy 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 a 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
```

### 8.6.2 | Deploy Multiple Files

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

#### 8.6.3 | Deploy 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**.

#### 8.6.4 | Deploy to a Virtual Repository

Artifactory supports deploying artifacts to a virtual repository.

# JFrog Artifactory Documentation

## Displayed in the header

To enable this, you must first designate one of the local repositories aggregated by the virtual repository as the default deployment target.

### Set the Default Deployment Repository using the UI

In the platform UI, the default deployment repository is defined on the virtual repository's configuration screen.

To set the default deployment repository in the UI:

1. In the **Administration** module, select **Repositories**.
2. Click the **Virtual** tab.
3. Locate the relevant virtual repository and click its repository key.
4. Scroll down to the **Default Deployment Repository** field and select one of the local repositories from the drop-down list.



5. Click **Save**.

### Set the Default Deployment Repository using the API

You can set the Default Deployment Repository using the `defaultDeploymentRepo` parameter of the Virtual Repository Configuration JSON used by the Update Repository Configuration REST API endpoint. Once the deployment target is configured, you may deploy artifacts 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.

#### Note

If you specify a **Default Deployment Repository** for a virtual repository, the corresponding Set Me Up dialog for the repository will include instructions and code snippets for deploying to that repository.

#### 8.6.5 | Deploy Large Files Using Multi-Part Upload

For large files, Artifactory implements a fast and reliable multi-part upload approach with the JFrog CLI. One of the main advantages of multi-part upload is that, in the case of upload failure, it has a retry mechanism that resumes uploads from the point of failure, thus preserving all content that was uploaded before the failure. In contrast, with the standard upload, an upload failure will result in the loss of all data and require a restart from the beginning.

The JFrog CLI automatically uses multi-part upload for large files without any user intervention, according to the values of the `--min-split`, `--split-count`, and `--chunk-size` settings.

- The `--min-split` setting determines the minimum file size required for multi-part upload. Its default value is 200 MB.
- The `--split-count` setting determines the number of parts that can be concurrently uploaded per file during a multi-part upload. Its default value is 5.
- The `--chunk-size` setting determines the upload chunk size in MB of the parts that are concurrently uploaded. Its default value is 20.

You can change the values for `--min-split`, `--split-count`, and `--chunk-size` when you run an upload command in the CLI. For example:

```
jf rt upload filename.zip /target-repo --min-split=150 --split-count=6 --chunk-size=25
```

The default values for `--min-split`, `--split-count`, and `--chunk-size` can be changed per command, but after running the command the values for these settings revert to the default. For more information on uploading files with the JFrog CLI, click [here](#).

#### Note

Multi-part Upload is available on Artifactory version 7.90.7 and later versions, and using JFrog CLI version 2.62.2 and later versions.

### Configuring Multi-part Upload for Self-Hosted Platforms

Multi-part upload is available on Self-Hosted platforms using S3 Storage or Google Cloud Storage configured in `binarystore.xml`.

#### Note

Multi-part upload is not supported on Self-Hosted platforms with S3-Sharding or when S3 storage is configured with Client-side KMS encryption (`kmsClientSideEncryptionKeyId`).

To enable multi-part upload on Self-Hosted platforms, set the Artifactory system property `artifactory.multipart.upload.enabled = true`. For more information, see Artifactory System Properties.

#### Multi-part Upload with Encryption using a KMS Key

To perform multi-part upload with encryption using a KMS key, you must have permission for the `kms:Decrypt` and `kms:GenerateDataKey` actions on the key. These permissions are required because Amazon S3 must decrypt and read data from the encrypted file parts before it completes the multi-part upload.

If your IAM user or role is in the same AWS account as the KMS key, then you must have these permissions on the key policy. If your IAM user or role is not in the AWS account as the KMS key, then you must have permissions on both the key policy and your IAM user or role.

#### How does it work?

The diagram below shows the sequence of the events that occur when a user performs a multi-part upload.

1. The user employs the JFrog CLI to submit an upload request.
2. Artifactory authenticates the request.
3. JFrog CLI splits the file into parts and sends a request to Artifactory for five pre-signed URLs for the first five parts.

#### Note

The default number of parts that can be uploaded concurrently is five, but this number can be modified with the `--split-count` variable in the JFrog CLI.

4. Artifactory sends the pre-signed URLs to JFrog CLI.
5. JFrog CLI concurrently uploads the first five parts directly to the S3 bucket.
6. Steps 3-5 are repeated until all parts are uploaded.
7. When all parts have been uploaded, JFrog CLI sends a 'Complete Upload' trigger to the S3 bucket.
8. The S3 bucket merges the parts into a single file (this merge can take some time). While the merge is happening, JFrog CLI polls Artifactory regarding the progress of the merge.
9. Artifactory verifies the checksum and notifies JFrog CLI that the upload has completed successfully.

### 8.6.6 | 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.

### 8.7 | Filtered Resources

*Requires a Pro license.*

The Filtered Resources 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.

# JFrog Artifactory Documentation

## Displayed in the header

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.

### 8.7.1 | Mark 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**.



### 8.7.2 | 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

### 8.7.3 | Provision 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. From the **Application module**, go to **Artifactory | Artifacts** and select a Maven, Gradle or Ivy package and click **Set Me Up** to display the settings generator.
2. Select your build tool, set the appropriate repositories and click **Generate Settings**.



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

:

#### 8.7.4 | Provisioning 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>
```

## 8.8 | Using WebDAV

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.

### 8.8.1 | 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 to 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=="
```

### 8.8.2 | 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.

## 8.9 | Artifactory Query Language

**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. AQL can only extract data that resides in your instance of Artifactory, so it runs on **Local Repositories, Remote Repositories Caches, and Virtual Repositories**.

From Artifactory 7.17.4, you can search within remote 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":{"$eq":"Main.class"} , "archive.entry.path":{"$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"}
})
```

# JFrog Artifactory Documentation

## Displayed in the header

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"
      } ]
    }, {
      "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"
      } ]
    }, {
      "entry.name" : "MANIFEST.MF",
      "entry.path" : "META-INF"
    } ]
  }, {
    "range" : {
      "start_pos" : 0,
      "end_pos" : 2,
      "total" : 2
    }
  }
]
```

For detailed information about AQL, refer to the following topics:

- [AQL Architecture](#)
- [AQL Syntax](#)
- [Using Fields in AQL](#)
- [AQL Execution](#)
- [AQL Entities and Fields](#)
- [Search Criteria Construction](#)
- [Specify Output Fields](#)
- [Sorting in AQL](#)
- [Display Limits and Pagination](#)
- [Using AQL With Remote Repositories](#)
- [Using AQL With Virtual Repositories](#)

# JFrog Artifactory Documentation

## Displayed in the header

### 8.9.1 | AQL 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.

The following primary domains are supported:

Domain	Form of Query
Item	items.find(...)
Build	builds.find(...)
Entry	archive.entries.find(...)
Promotion	build.promotions.find(...)
Release	releases.find(...)
	<b>Note</b> To run queries on the build domain, you must be an admin user.
	archive.entries.find(...)
	build.promotions.find(...)
	releases.find(...)
	<b>Note</b> The Release domain refers to Release Bundles v1, which is created in JFrog Distribution, and not the Release Bundles v2 created in Artifactory. For more information, see <a href="#">Types of Release Bundles</a> .

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.

### 8.9.1.1 | AQL 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	6.0.0/ 7.0.0
Item	✓	✓	✓	✓
Item.Property	✓	✓	✓	✓
Statistic	✓	✓	✓	✓
Archive	✗	✓	✓	✓

	3.5.0	4.2.0	4.7.0	6.0.0/ 7.0.0
Archive.Entry	✗	✓	✓	✓
Artifact	✗	✓	✓	✓
Dependency	✗	✓	✓	✓
Module	✗	✓	✓	✓
Module.Property	✗	✓	✓	✓
Build	✗	✓	✓	✓
Build.Property	✗	✓	✓	✓
Promotion	✗	✗	✓	✓
Release	✗	✗	✗	✓
Release_artifact	✗	✗	✗	✓

#### Note

- To run queries on the build domain, you must be an admin user.
- The Release domain refers to Release Bundles v1, which is created in JFrog Distribution, and not the Release Bundles v2 created in Artifactory. For more information, see [Types of Release Bundles](#).

#### 8.9.2 | AQL Syntax

```
<domain_query>.find(<criteria>).include(<fields>).sort(<order_and_fields>).offset(<offset_records>).limit(<num_records>).distinct(<true|false>)
```

The following table describes the elements.

Element	Description
domain_query	The query corresponding to the primary domain. Must be one of <b>items</b> , <b>builds</b> or <b>entries</b> .
criteria	The search criteria in valid JSON format
fields	(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
order_and_fields	(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.
num_records	(Optional) The maximum number of records that should be extracted. If omitted, all records answering the query criteria will be extracted.
offset	(Optional) The offset from the first record from which to display results (i.e. how many results should be skipped for display)
distinct	(Optional) Disable or enable unique query results. Enabled by <b>default</b> .

From Artifactory 7.37.x, the AQL query supports disabling of unique results. This can be used to speed up the query or to get the results "as is". This replaces the previous search results for text-based AQL queries, which are unique by default (native SQL selects contain 'distinct' terms).

For example:

```
// Find all unique jars's names in artifactory
items.find({"name" : {"$match":"*.jar"}}).include("name")
```

# JFrog Artifactory Documentation

## Displayed in the header

```
// Find all jars's names in artifactory includes repeated values across different repositories.  
items.find({"name" : {"$match": "*.jar"}}).include("name").distinct(false)  
  
// Find all jar items in a repository called "my_local".  
// Property "path" already unique inside single repository and unique results can be disabled for performance reason.  
items.find(  
    {"repo" : "my_local"},  
    {"name" : {"$match": "*.jar"}}  
)  
.include("path")  
.distinct(false)
```

### 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")
```

### 8.9.3 | Using Fields in AQL

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"})
```

### 8.9.4 | AQL Execution

To execute an AQL query, use the Artifactory Query Language (AQL) REST API.

#### Artifactory Query Language (AQL)

**Description:** Flexible and high performance search using Artifactory Query Language.

**Since:** 3.5.0

**Security:** Requires an authenticated user. Certain domains/queries may require Admin access.

**Usage:** POST /api/search/aql

**Consumes:** text/plain

#### Note

For information on limiting the maximum number of AQL search results, click [here](#).

#### 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  
        }  
    ]  
}
```

# JFrog Artifactory Documentation

## Displayed in the header

```
"end_pos" : 1,  
"total" : 1,  
"limit": 5,  
"notification": "AQL query reached the search hard limit, results are trimmed."  
}  
}
```

### 8.9.5 | AQL Entities and Fields

You may issue a `find` request according to the syntax, 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
item	path	String	The full path associated with this item  <b>Note</b> The path for folder should not include a /*, whereas for items, it is required.
item	name	String	The name of the item
item	created	Date	The date when the item was created
item	modified	Date	File system timestamp indicating when the item was last modified
item	updated	Date	When the item was last uploaded to a repository
item	created_by	String	The name of the item owner
item	modified_by	String	The name of the last user that modified the item
item	type	Enum	The item type (file/folder/any). If type is not specified in the query, the default type searched for is file
item	depth	Int	The depth of the item in the path from the root folder
item	original_md5	String	The item's md5 hash code when it was originally uploaded
item	actual_md5	String	The item's current md5 hash code
item	original_sha1	String	The item's sha1 hash code when it was originally uploaded
item	actual_sha1	String	The item's current sha1 hash code
item	sha256	String	The item's sha256 hash code  <b>SHA-256 is supported from Artifactory version 5.5</b>  You can use AQL search only on artifacts deployed to Artifactory version 5.5 or above, or if you have migrated your database as described in SHA-256 Support after upgrading Artifactory to version 5.5 and above.
item	size	Long	The item's size on disk
item	virtual_repos	String	The virtual repositories which contain the repository in which this item is stored.

# JFrog Artifactory Documentation

## Displayed in the header

Domain	Field Name	Type	Description
archive	Note		<b>Enable additional archive types</b>  The archive domain currently contains no fields  To enable search for specific archive type, the `index=true` should be specified in the mimetypes.xml file for this archive type. See Configuration Files: MIME Type
entry	name	String	The entry name
entry	path	String	The path of the entry within the repository
promotion	created	String	When the build was promoted
promotion	created_by	String	The Artifactory user that promoted the build
promotion	status	String	The status of the promotion
promotion	repo	String	The name of the repository to which the build was promoted
promotion	comment	String	A free text comment about the promotion
promotion	user	String	The CI server user that promoted the build
build	url	String	The URL of the build
build	name	String	The build name
build	number	String	The build number
build	created	Date	File system timestamp indicating when the item was last modified
build	created_by	String	The name of the user who created the build
build	modified	Date	File system timestamp indicating when the build was last modified
build	modified_by	String	The name of the last user that modified the build
build	started	Date	The build start time. The value for this field is directly taken from the relevant build's build-info.json file  The field is immutable, and does not change upon build promotion or build replication
property	key	String	The property key
property	value	String	The property value
stat	downloaded	Date	The last time an item was downloaded
stat	downloads	Int	The total number of downloads for an item
stat	downloaded_by	String	The name of the last user to download this item
stat	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

# JFrog Artifactory Documentation

## Displayed in the header

Domain	Field Name	Type	Description
stat	remote_downloaded	Date	The last time an item was downloaded from a smart remote repository proxying the local repository in which the item resides
stat	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
stat	remote_origin	String	The address of the remote Artifactory instance along a smart remote proxy chain from which the download request originated.
stat	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
artifact	type	String	The type of the artifact
artifact	sha1	String	The SHA1 hash code of the artifact
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
dependency	scope	String	The scope of the dependency
dependency	type	String	The type of the dependency
dependency	sha1	String	The SHA1 hash code of the dependency
dependency	md5	String	The MD5 hash code of the dependency
release	name	String	The name of the release bundle
release	version	String	The version of the release bundle
release	status	String	The status of the release bundle
release	created	String	The date when the release bundle was created
release	signature	String	The release bundle signature
release_artifact	path	String	The release artifact full repo path

### 8.9.6 | Search Criteria Construction

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.

#### 8.9.6.1 | Field Criteria

The general way to specify a criterion on a field is as follows:

# JFrog Artifactory Documentation

## Displayed in the header

```
{"<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>"}  
Element Description  
Example Find items whose "name" field equals "ant-1.9.4.jar"  
Regular notation items.find({"name": {"$eq": "ant-1.9.4.jar"}})  
Short notation items.find({"name": "ant-1.9.4.jar"})
```

### 8.9.6.2 | 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>"}  
Element Description  
Example Find items with associated properties named "license" with a value that equals "GPL"
```

# JFrog Artifactory Documentation

## Displayed in the header

Element	Description
Regular notation	<code>items.find({"@artifactory.licenses" : {"\$eq" : "GPL"}})</code>
Short notation	<code>items.find("@artifactory.licenses" : "GPL")</code>

### 8.9.6.3 | 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  
`<criterion>={<$and|$or>:[{<criterion>},{<criterion>}]}`

#### 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 maven-remote or my-local repositories.  
items.find({"type" : "file","$or": [{"repo" : "maven-remote", "repo" : "my-local"}]})  
  
//Find all the items that were created in a build called "my_debian_build" and whose name ends with ".deb" or all items created in a build called "my_yum_build" 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*"}})
```

### 8.9.6.4 | Match 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": {"$match": "*GPL*"},
    "@license": {"$nmatch": "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": {"\$nmatch": "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": {"\$nmatch": "LGPL-2.1\*"}.

If we use the **\$msp** operator as follows:

```
items.find({
    "$msp": [
        "@license": {"$match": "*GPL*"},
        "@license": {"$nmatch": "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": {"\$nmatch": "LGPL-2.1\*"}} criterion.

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": {"\$nmatch": "LGPL-2.1\*"}} criterion.

#### Tip

Note that the \$msp operator works equally well on all domains that have properties: **item**, **module** and **build**.

##### 8.9.6.5 | Comparison Operators

The following table lists the full set of comparison operators.

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.

##### 8.9.6.6 | Using Wildcards

# JFrog Artifactory Documentation

## Displayed in the header

To enable search using non-specific criteria, AQL supports wildcards in common search functions.

### Use 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

Element	Description
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>

Element	Description
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 of Wildcard Usage

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 "GPL"
<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 "*GPL*"

#### 8.9.6.7 | Date and Time Format in AQL

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)

# JFrog Artifactory Documentation

## Displayed in the header

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.

### 8.9.6.8 | Relative Time Operators in AQL

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:

Operator	Description
\$before	The query is run over complete period up to specified time.
\$last	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:

Time Period	Suffix
milliseconds	mills, ms
seconds	seconds, s
minutes	minutes
days	days, d
weeks	weeks, w
months	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"}})
```

### 8.9.7 | Specify 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.

#### 8.9.7.1 | Display All Fields

Use: `.include("*")`

For example:

```
//Find all items, and display all the item fields
items.find().include("*")
```

# JFrog Artifactory Documentation

## Displayed in the header

### 8.9.7.2 | Display 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
    }
}
```

### 8.9.7.3 | 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.

# JFrog Artifactory Documentation Displayed in the header

## 8.9.7.4 | Filtering Properties by Key

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. However, note that `.include` will also filter out items that do not have this property.

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")
```

## 8.9.8 | Sorting in AQL

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>,... ]"})
```

### Note

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"]})
```

### Limitation

The `.sort` element has the following limitation:

- If your query has an `include` element, you can only specify fields from the primary domain in it.

For example, in the following query, `.sort` will not work because the primary domain is an `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")
```

This means that if you search for an item and include the property you will not be able to sort by the property.

```
items.find({"repo": "example-repo-local"}).include("repo", "path", "name", "created", "@build.number")
```

## 8.9.9 | Display Limits and Pagination

### 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 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)
```

## 8.9.10 | Using AQL With Remote Repositories

### Searching Remote Repositories

From Artifactory version 7.17.4, you can search directly within remote or virtual repositories. This requires providing credentials to the remote Artifactory on the remote repository configuration under the Advanced tab.

To enable a search within a remote repository, add the transitive tag to the search query.

# JFrog Artifactory Documentation

## Displayed in the header

```
items.find("repo": "remote-repo").transitive()
```

### Rules and Guidelines for Using AQL in Remote Repositories

The following rules and guidelines apply when searching remote and virtual repositories:

- The remote repository MUST be an Artifactory instance.
- The query MUST include a single repository and a \$eq clause.

#### Example

```
items.find("repo": "remote-repo").transitive()
```

- You cannot use Sort and Offset flags within the AQL Query.
- The primary domain can only contain Items.
- Include may only have the following domains: Items and ItemProperties.
- The search will run on the first five remote repositories within the virtual repository.

8.9.10.1 | Examples of Searching in Remote Repositories with AQL

#### Example 1: Input

The following example searches for all the fields of the domain item.

```
items.find(
  {
    "repo" : "docker-remote-repo"
  }
).transitive()
```

#### Example 1: Output

```
{
  "results": [
    {
      "repo": "docker-remote-repo-cache",
      "path": "alpine/latest",
      "name": "manifest.json",
      "type": "file",
      "size": 528,
      "created": "2021-03-21T13:54:52.383+02:00",
      "created_by": "admin",
      "modified": "2021-03-21T13:54:32.000+02:00",
      "modified_by": "admin",
      "updated": "2021-03-21T13:54:52.384+02:00"
    },
    ...
  ],
  "range": {
    "start_pos": 0,
    "end_pos": 12,
    "total": 12
  }
}
```

#### Example 2: Input

The following example limits the searches to results listed in the includesection.

```
items.find(
  {
    "repo" : "docker-remote-repo"
  }
).include("name").transitive()
```

#### Example 2: Output

```
{
  "results": [
    {
      "name": "manifest.json"
    },
    {
      "name": "sha256__4c0d98bf9879488e0407f897d9dd4bf758555a78e39675e72b5124ccf12c2580"
    },
    {
      "name": "sha256__e50c909a8df2b7c8b92a6e8730e210ebe98e5082871e66edd8ef4d90838cbd25.marker"
    },
    {
      "repo": "docker-remote-repo",
      "name": "manifest.json"
    },
    {
      "repo": "docker-remote-repo",
      "name": "repository.catalog"
    },
    {
      "repo": "docker-remote-repo",
      "name": "sha256__4c0d98bf9879488e0407f897d9dd4bf758555a78e39675e72b5124ccf12c2580"
    },
  ],
}
```

```
{  
    "repo": "docker-remote-repo",  
    "name": "sha256_e50c909a8df2b7c8b92a6e8730e210ebe98e5082871e66edd8ef4d90838cbd25"  
}  
],  
"range": {  
    "start_pos": 0,  
    "end_pos": 7,  
    "total": 7  
}  
}
```

### 8.9.11 | Using AQL With Virtual Repositories

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.

### 8.9.12 | Limiting AQL Search Results

The maximum number of AQL search results is limited as follows:

- For Artifactory Cloud platforms starting from version 7.100.2, a fixed value of 500,000 is set as the default maximal result set size to limit the search/AQL API call response. When this limit is reached, no further results are displayed and the user receives the following message: "AQL query reached the search hard limit, results are trimmed."
- For Artifactory Self-Hosted platforms, by default there is no maximum limit. However, from Self-Hosted version 7.90.6 and later users can limit search results with the system property `aql.search.query.max.limit`.

## 9 | JFrog Container Registry

The JFrog Container Registry is a repository manager, which supports Docker and Helm registries and Generic repositories, allowing you to build, deploy and manage your container images while providing powerful features with fine-grained permission control behind a sleek and easy-to-use UI.

JFrog Container Registry is available as a self-hosted (Freemium) or SaaS solution and supports private and public Docker images, Helm Charts, and Generic repositories. The internal permission mechanism helps DevOps teams decide who can access what with fine-grained access control. It provides visibility and management of your Images with advanced querying based on metadata and allows you to run multiple registries per instance/account.

JFrog Container Registry comes with an Easy to Use UI with an Advanced Image Layer View and images search capabilities. A powerful REST API and CLI can be used to push, pull, and easily manage your images.

# JFrog Artifactory Documentation

## Displayed in the header

### Key Features

- **Hybrid and Multi-Cloud Environments:** You can host JFrog Container Registry on your own infrastructure, in the Cloud, or use the SaaS solution providing maximum flexibility and choice.
- **Dedicated Docker Registry:** You can set up a secure private Docker registry in minutes to manage all your Docker images while exercising fine-grained access control. JFrog Container Registry 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 registries such as the Docker Hub. To start configuring your Docker registry, see Configuring Docker Repositories.
- **Dedicated Helm Registry:** The Helm package search in JFrog Container Registry is customized to allow users to search for Helm repositories by "App version" and not only by "Version", which refers to the Chart version. App Version is a useful piece of information as it lets your users know what version of your app they are using, as the chart version could differ. You can search for the parameter after you add it to the *Chart.yaml* file. For more information, see Helm Registry.
- **Generic Repositories:** JFrog Container Registry supports Generic repositories that are not associated with any particular package type and can be used to upload packages in any format. Generic repositories do not maintain separate package indexes, because they are not specific to any package type. They are useful when you want to proxy unsupported package types, store installers, navigation files, audio files, etc.
- **Local, Remote and Virtual Repositories:** You can start managing your container images by setting up **Local repositories** that are physical, locally-managed repositories. Typically these are used to deploy internal and external releases as well as development builds, but they can also be used to store images that are not widely available on public repositories such as 3rd party commercial images.
  - **Remote repositories** allow you to set up a caching proxy for external registries such as the Docker Hub. Artifacts are stored and updated in remote repositories according to various configuration parameters that control the caching and proxying behavior. You can even set up your remote repository as a **Smart Remote Repository** to proxy a local or remote repository from a different JFrog Container Registry, essentially caching all distant repository content inside your own JFrog Container Registry instance. Smart remote repositories are especially useful when changes are made to the original container image, e.g. when its properties are changed or when it is deleted.
  - **Virtual repositories** encapsulate any number of local and remote repositories and represents them as a unified repository accessed from a single URL. It gives you a way to manage which repositories are accessed by developers since you have the freedom to mix, match and modify the actual repositories included within the virtual repository.
- **Multiple Container Registries:** Useful for separating teams/projects and for promoting images from one environment to the next (Development, Staging, and Production).

### 9.1 | JFrog Container Registry Documentation

The JFrog Container Registry is powered by JFrog Artifactory with a set of features that have been customized to serve the primary purpose of running Docker and Helm packages in a Container Registry.

As JFrog Container Registry is a version of Artifactory, refer to the JFrog Artifactory documentation when working with JFrog Container Registry. For example, Configuring Artifactory, Installing Artifactory, Upgrading Artifactory, and Artifactory Release Notes.

#### Tip: View All Pages Relevant to JFrog Container Registry

To view the full list of pages supported by JFrog Container Registry, type *jfrog\_container\_registry* in the search bar in the left panel.

To view a list of supported JFrog Container Registry features, see the JCR Functionality Table and to compare JFrog Container Registry with other versions, see the Self-Hosted JFrog Platform Comparison Matrix or the Cloud JFrog Platform Comparison Matrix depending on your needs.

[Try JFrog Container Registry](#)

[Integration Benefits](#)

[Container Registry](#)

### 9.2 | Container Registry Functionality

The following table lists the JFrog Container Registry functionality and supported tools.

Category	Feature
Basic artifact management	Proxy and cache remote repository artifact
	Include/exclude patterns for stored artifacts
	Deploy artifacts via the UI or via REST/HTTP
	Move/copy/delete artifacts through the UI
	Checksum-based Storage with Deduplication
	Promotion, demotion and cleanup of build artifacts
Build management	

# JFrog Artifactory Documentation

## Displayed in the header

Category	Feature
Working with CI servers	Managing build artifacts for reproducible builds Garbage Collection Integration with all leading CI-servers
Security	LDAP Authentication Role-based authorization with teams and permissions Active Directory
Manipulating metadata	Search by Name, Archive, Property or Checksum Values Annotate Artifacts with Searchable Properties
Developer tools	Artifactory Query Language (AQL) Powerful REST API for Release Automation Integrated with JFrog CLI
Self-hosted advanced storage system	S3 Object Storage Incremental and Historical Backup Services
Cloud dedicated features	SaaS-based Maintenance-free Hosted Repository Always up-to-date JFrog Container Registry Version Setup Free Automated Backups
Free Upgrades	

### 9.3 | Get Started: JFrog Container Registry

You can host the JFrog Container Registry as a Self-hosted (Freemium) version on your own infrastructure or use the Cloud version hosted by JFrog. Both versions of JFrog Container Registry share an almost identical set of features, look and feel, and functionality but provide you with the flexibility you need when it comes to meeting your organization's hosting and infrastructure

# JFrog Artifactory Documentation

## Displayed in the header

requirements.

After you have determined which version best accommodates your needs, you can proceed to get started.

- Get Started with the Self-hosted Version
- Get Started with the Cloud Version

### 9.3.1 | Set Up JFrog Container Registry

This section reviews how to set up the JFrog Container Registry for the following two deployment types:

- Set up JFrog Container Registry Self-hosted Version
- Set up JFrog Container Registry Cloud Version

#### 9.3.1.1 | Set up JFrog Container Registry Self-hosted Version

##### The JFrog Container Registry is powered by JFrog Artifactory

The JFrog Container Registry is powered by JFrog Artifactory with a set of features that have been customized to serve the primary purpose of running Docker and Helm packages in a Container Registry. Refer to the JFrog Artifactory documentation when working with JFrog Container Registry. For example, Setting Up Artifactory , Installing Artifactory , Upgrading Artifactory , Artifactory REST API and Artifactory Release Notes.

Please note that the JFrog Container Registry installation and configuration paths and code examples refer to *artifactory* since JFrog Container Registry is powered by JFrog Artifactory.

1. Go to the [JFrog Container Registry Download](#) page.
2. Download your preferred installation method: Standalone (Zip), RPM, Debian or Docker or Helm Charts.
3. Proceed to install and configure JFrog Container Registry according to the JFrog Artifactory instructions in this guide.
4. To get up and running quickly, follow the Onboarding Wizard that is invoked the first time you start JFrog Container Registry.

#### 9.3.1.2 | Set up JFrog Container Registry Cloud Version

1. Go to [JFrog Container Registry](#) page.
  2. Scroll to the bottom of the page and click Cloud **Try Free Now**.
- The Cloud registration form opens.
3. Follow the Onboarding wizard to select your preferred cloud provider and fill in the registration form.
  4. Click **Next**. A Thank You page is displayed confirming the registration process has completed successfully.
- An email containing your JFrog Container Registry login information and credentials will be sent to you.
5. To get up and running quickly, follow the Onboarding Wizard that is invoked the first time you start JFrog Container Registry.

### 9.3.2 | Run the Onboarding Wizard

Get up and running quickly and easily with your new JFrog Container Registry installation using the onboarding wizard. The wizard is launched when you install the JFrog Container Registry, making sure you can complete the set up with the minimal information needed to get started.

Starting the JFrog Container Registry for the first time launches the JFrog Container Registry onboarding wizard.

The initial setup lets you configure the following basic settings:

- Proxy server (Self-hosted only)
- Initial default repositories

#### Welcome

The beginning of the onboarding wizard. Click **Get Started** to continue.

**Step 1: EULA**

Accept the EULA and click **Next** to continue.

**Step 2: Subscribe to Newsletter**

Subscribe for updates by entering one or more email addresses. You can unsubscribe at anytime.

Click **Next** to continue.

**Step 3: Reset Admin Password**

When the JFrog Container Registry is installed, it creates a default user with admin privileges predefined in the system. In this step, you must reset the admin password by entering a new password.

Click **Next** to continue.

**Step 4: Set Base URL**

Set the base URL for accessing JFrog. For more information, see General System Settings.

Click **Next** to continue.

**Step 5: Configure Default Proxy**

*\*For Self-hosted only*

Configure a proxy server. Configuring a default proxy is optional, so you can skip this step. However, if you do decide to configure the default proxy now, all three fields, Proxy Key, Host, and Port, are mandatory.

Click **Next** to continue.

**Step 6: Create Repositories**

Select one or more of the Docker, Helm, or Generic package formats for which JFrog Container Registry should create default repositories.

Click **Next** to continue.

## Step 7: Summary

Review the default repositories created according to your selection. Click **Finish** to complete the wizard and get started with JFrog Container Registry.

### 9.3.3 | Get Familiar with Your JFrog Container Registry

To learn more about working with your services:

- Use the JFrog Platform Administration Documentation to learn about administrative tasks in the UI
- Use the JFrog Artifactory Documentation to learn about application-level tasks in the UI for both administrators and end-users
- Use the REST API guide for end-user information, as well as the [Artifactory Query Language](#) and working with builds.

### 9.3.4 | Upgrade JFrog Container Registry

The procedure for upgrading JFrog Container Registry to a higher version or to Artifactory Pro is identical to upgrading JFrog Artifactory. For more information, see the [Upgrade Procedure](#).