Repository Management

Overview

The JFrog Platform hosts the following repository types:

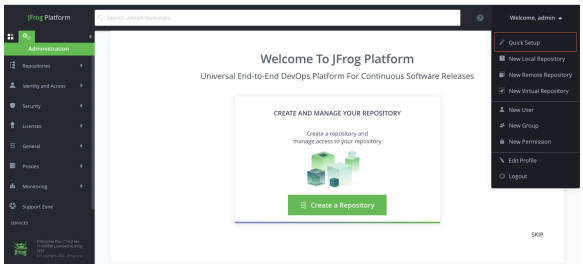
- Local
- Remote
- Virtual
- Federated
- Distribution
    - Release Bundle Repository

Local and remote repositories are true physical repositories, while a virtual repository is actually an aggregation of them used to create controlled domains for search and resolution of artifacts.

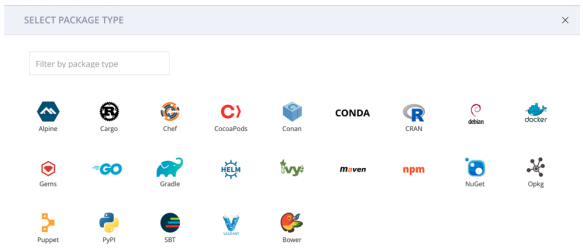
To manage repositories, go to **Repositories** under the **Administration.**

You can also use the Quick Setup, 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.

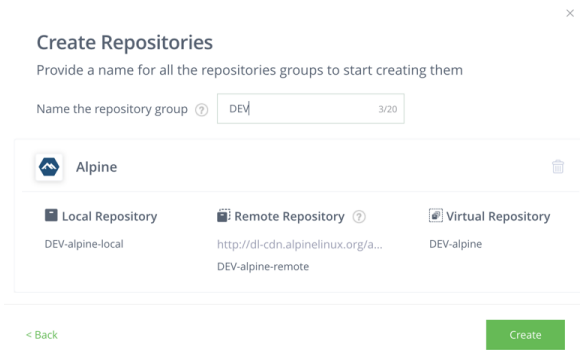
Step 1 From the user drop-down menu, select Quick Setup.



Step 2 Select one or more package types to create the default repositories.

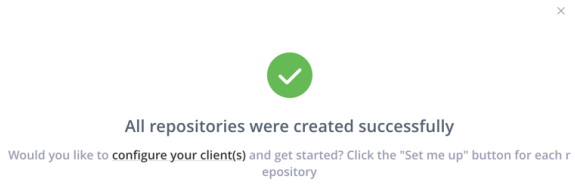


Step 3 Provide a name for the group of repositories. The name will be used as a prefix that will help you to manage the repositories.



Step 4 Click Create.

Page Contents



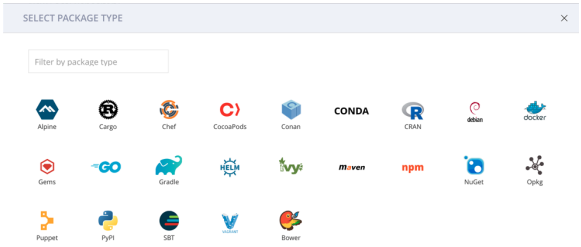
You have completed creating your repositories, you can continue to configure your clients, and deploy artifacts, as described in Package Management.

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 and can not 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.



Generic Repositories

You can define a repository as **Generic** in which case it has no particular type, and you may upload packages of any type. Generic repositories do not maintain separate package indexes. For using a client associated with a specific package type (e.g. yum, gem) you should create a matching repository. 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.

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 packages you upload. If you do upload packages of the wrong type to a repository, Artifactory will not index the package or update the metadata for the repository.

Local Repositories

Local repositories are physical, locally-managed repositories into which you can deploy artifacts.
Artifacts in a local repository can be accessed directly using the following URL:

```
http://<host>:<port>/artifactory/<local-repository-name>/<artifact-path>
```

For full details on configuring local repositories, please refer to the Local Repositories page.

Remote Repositories

A remote repository serves as a caching proxy for a repository managed at a remote URL (which may itself be another Artifactory remote repository). Artifacts are stored and updated in remote repositories according to various configuration parameters that control the caching and proxying behavior. You can remove artifacts from a remote repository cache but you cannot actually deploy a new artifact into a remote repository.

Artifacts in a remote repository can be accessed directly using the following URL:

`http://<host>:<port>/artifactory/<remote-repository-name>/<artifact-path>`

This URL will fetch a remote artifact to the cache if it has not yet been stored.

In some cases it is useful to directly access artifacts that are already stored in the cache (for example to avoid remote update checks).

To directly access artifacts that are already stored in the cache you can use the following URL:

`http://<host>:<port>/artifactory/<remote-repository-name>-cache/<artifact-path>`

Proxy vs. Mirror

A remote repository acts as a **proxy** not as a mirror. Artifacts are not pre-fetched to a remote repository cache. They are only fetched and stored *on demand* when requested by a client.
Therefore, a remote repository should not contain any artifacts in its cache immediately after creation. Artifacts will only be fetched to the cache once clients start working with the remote repository and issuing requests.

For full details on configuring remote repositories please refer to the Remote Repositories page.

Virtual Repositories

A virtual repository (or "repository group") aggregates several repositories with the same package type under a common URL.

Generic Virtual Repositories

By their nature, Virtual Repositories whose package type has been specified as **Generic** can aggregate repositories of any type, however generic virtual repositories do not maintain any metadata.

The Default Virtual Repository (Deprecated)

Artifactory offers an option to use a global virtual, which contains all local and remote repositories.

By default this option is disabled, to enable the Default Virtual Repository edit the `artifactory.system.properties` located at `$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 restart your Artifactory service.

Once enabled the repository is available at:

`http://<hostname>:<port>/artifactory/repo`

Virtual Resolution Order

When an artifact is requested from a virtual repository, the order in which repositories are searched or resolved is local repositories first, then remote repository caches, and finally remote repositories themselves.

Within each of these, the order by which repositories are queried is determined by the order in which they are listed in the configuration as described in General Resolution Order below.

For a virtual repository, you can see the effective search and resolution order in the **Included Repositories** list view in the **Basic** settings tab. This is particularly helpful when nesting virtual repositories. For more details on configuring a virtual repository please refer to the Virtual Repositories page.

Federated Repositories

From JFrog Artifactory 7.18.3, the JFrog Platform enables you to create Federated repositories which allow mirroring of artifacts and their metadata with other repositories of a Federated type located on remote JFrog Deployments (JPDs) in a multisite environment. The Federated repository functions similarly to a local repository on the JPD, but is grouped together logically with other Federated members located on other JPDs, to create a Federation. For more information, see Federated Repositories.

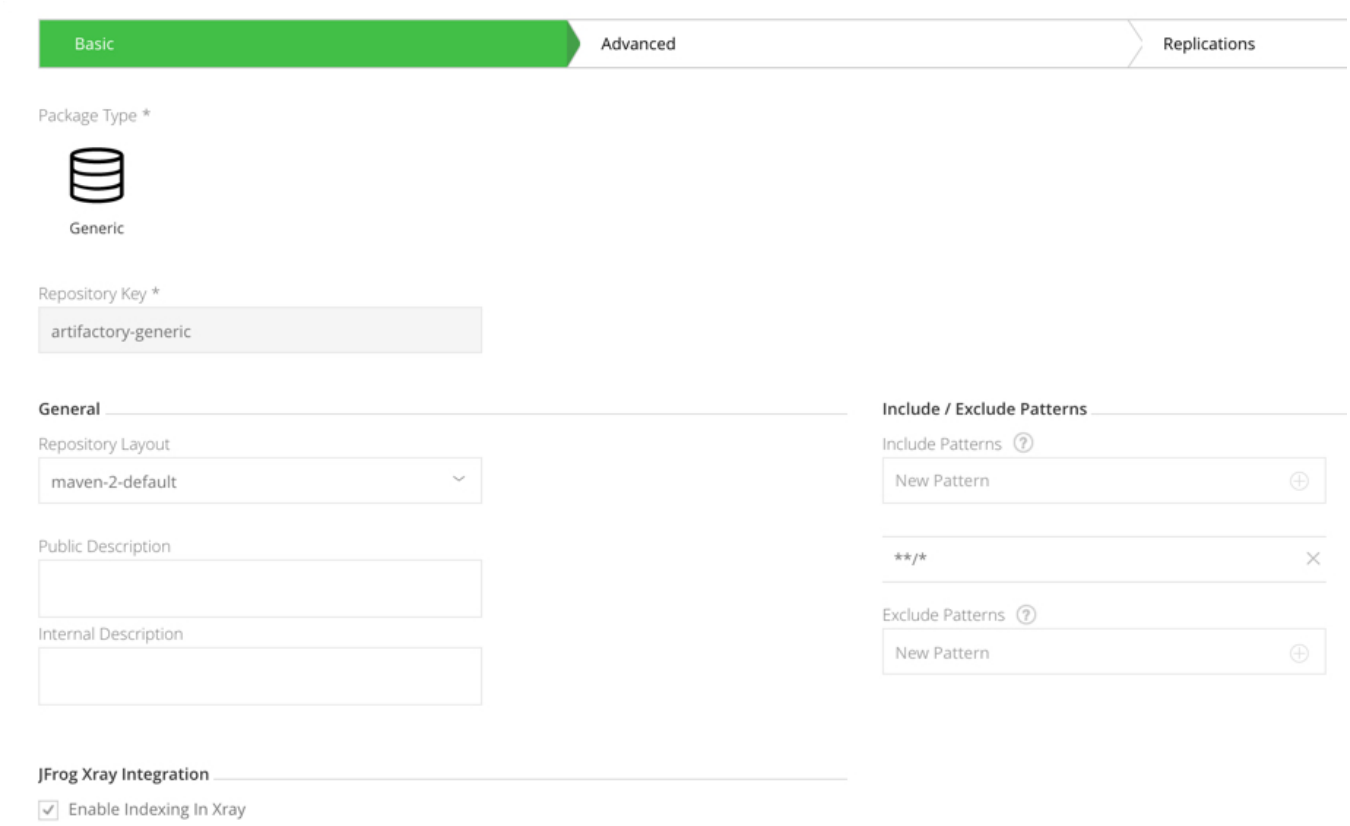
General Resolution Order

You can set the order in which repositories of each type (local, remote and virtual) are searched and resolved by simply ordering them accordingly within the corresponding section of the **Configure Repositories** page. To set the order you need to add the repositories to the list of selected repositories in the order in which they should be searched to resolve artifacts.

The order in which repositories are searched is also affected by additional factors such as security privileges, include/exclude patterns and policies for handling snapshots and releases.

Common Settings

Several of the settings are common for local, remote and virtual repositories. These are found in the **Basic** tab of the corresponding **New/Edit** screen. Additional settings may be found in the type-specific section according to the package types specified for the repository.



Package Type	The Package Type. This must be specified when the repository is created, and once set, cannot be changed.
Repository Key	The Repository Key. A mandatory identifier for the repository and must be unique. It cannot begin with a number or contain spaces or special characters. For local repositories, we recommend using a "-local" suffix (e.g. "libs-release-local").
Repository Layout	Sets the layout that the repository should use for storing and identifying modules. A recommended layout that corresponds to the package type defined is suggested, and index packages uploaded and calculate metadata accordingly.
Public Description	A free text field that describes the content and purpose of the repository.
Internal Description	A free text field to add additional notes about the repository. These are only visible to the administrator.

Include and Exclude Patterns	The **Include Patterns** and the **Exclude Patterns** fields provide a way to filter out specific repositories when trying to resolve the location of different artifacts.
	
	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:
	
	Consider that the Include Patterns and Exclude Patterns for a repository are as follows:
	
	`Include Patterns: org/apache/**,com/acme/**`
	`Exclude Patterns: com/acme/exp-project/**`
	
	In this case, 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.

Enable Indexing in Xray	Enables indexing on the repository for security and compliance analysis. Available with JFrog Xray.

Avoiding Security Risks with an Exclude Pattern

This section explains how to use Exclude Patterns to avoid the following security risks.

Prevent Exposure of Internal Artifacts Using Exclude Patterns

Any proprietary artifacts you deploy are stored within local repositories so that they are available for secured and authorized internal use. Anyone searching for one of your internal artifacts by name will extract it through Artifactory from the local repository.

However, consider what happens if a request for an internal artifact is inadvertently directed *outside* of the organization.

Two examples of how this could happen are:

- there is a simple typo in the requested artifact name
- the developer has requested a snapshot with a version number that does not exist.

In this case, since the system does not find the requested artifact in a local repository, it continues to search through the remote repositories defined in the system. A search through *all* the remote repositories defined in your system before returning "Not found".

This presents a security risk since any request made on a remote repository may be logged **exposing all details of the query including the full artifact name which may include sensitive business information**.

Best practices using an excludes pattern for remote repositories to avoid security risks like the Namespace Shadowing Attack

To avoid exposing sensitive business information as described above, we strongly recommend the following best practices:

- The list of remote repositories used in an organization should be managed under a single virtual repository to which all requests are directed
- All internal artifacts should be specified in the **Exclude Pattern** field of the virtual repository (or alternatively, of *each* remote repository) using wildcard characters to encapsulate the widest possible specification of internal artifacts.

Read more about scoped packages and exclude patterns >

Prevent Exposure of Internal Packages Using Exclude Patterns

Proxying a public remote repository that is not a trusted repository or is compromised can expose you to malicious artifacts. Sometimes these repositories allow anyone to deploy custom packages. For example, for npm, the public repository is `npmjs`, and anyone can deploy any version of any package he/she is the owner of. If a package does not have an owner (no one has previously deployed a version of it) anyone can deploy it and claim it.

For example, let's assume you have a library called "`almo-common-utils`" and its source is publicly accessible, if, for instance, it is bundled as part of publicly accessible products or web applications, it's written in Node and JFrog Artifactory has a set of remote (proxying the public repository), local (for sharing modules internally), and virtual repositories.

Consider the following:

- In a public repository, anyone can publish an unscoped library and call it whatever they want, i.e. "`almo-common-utils`" (unless there is a name conflict).
- There is no package named "`almo-common-utils`" in the public repository (because it's an internal corporate library), so there is no name conflict.
- You have projects declaring a dependency on "`almo-common-utils`" using a version range. For example, a range specifying the latest version compatible with a major version.

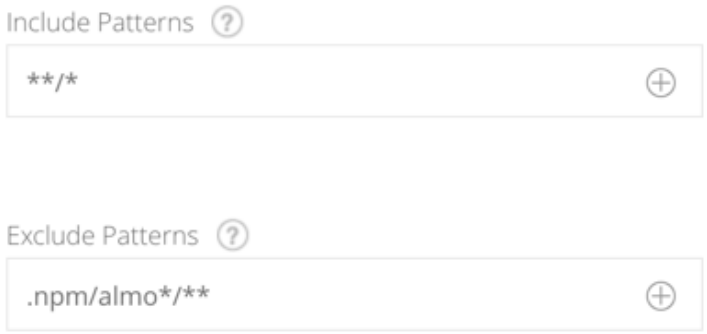
This presents a security risk, as an attacker can try to attack an unprotected organization by just having prior knowledge of the library "`almo-common-utils\"`", the major version of the library in use (let's say they know version 3 is used widely in the organization), and the content of the source code.

An attacker can clone and modify the source, embedding any malware inside, but still maintain compatibility with the original code, and upload it to the repository as "`almo-common-utils:3.99.99`". This will create a version update hijacking of an internal library, when "`almo-common-utils:^3.0.0`" is requested, the fake "`almo-common-utils`" from the repository is fetched.

To avoid exposing internal packages and internal packages version hijacking, we strongly recommend the following:

- **Use exclude patterns on your remote repositories.** Exclude the packages you do not want to search outside the organization in the remote repository. You can exclude by prefix (`.npm/almo*/**`), by scope (`.npm/@almo/*`) or by name (`.npm/almo-encoder/**`). By setting these exclude patterns on any public remote repository, you are effectively not merging those packages from the public repositories.
- **Only create and publish scoped packages.** Register an official organization for your company in the public repository to own a scope for your organization, and always publish only scoped packages. This also simplifies the exclude patterns, as you only need to exclude scoped packages to exclude all the packages from being searched in remote repositories.

  Here is an example of blocking packages with the scope "**almo"** from a remote repository.



Avoiding Performance Issues with an Include Pattern

Include patterns help you avoid clutter in your local repositories by making sure that only certain types of artifact can be hosted there. In a typical scenario, the system willl reference large all-purpose repositories such as JCenter or Maven Central for resolving artifacts.

In addition, Artifactory may reference any number of additional repositories which may host a more specialized and specific set of of artifacts.

If Artifactory receives a request for a deterministic set of artifacts (e.g. a specific version of an artifact), then it searches through the different repositories according to its resolution order until the artifact is found.

However, if Artifactory receives a request for a non-deterministic set of artifacts (e.g. all versions of *`maven-metadata.xml`*) then it must search through *all* of the repositories it references until it can provide a complete response.

In most cases, the majority of artifacts downloaded by an organization will come from one of the large all-purpose repositories, but in non-deterministic requests **performance is downgraded because Artifactory continues to search through all the specialized repositories** before it can return a response.

Best practices using an includes pattern for remote repositories to avoid needless and wasteful search

To avoid performing needless and wasteful search when responding to non-deterministic requests we strongly recommend that all specialized repositories be configured with an appropriate **Include Pattern** specifying only the set of artifacts that the organization might need.

In this case, non-deterministic requests for artifacts that are typically found in general purpose repositories will skip over the specialized repositories thereby improving performance.

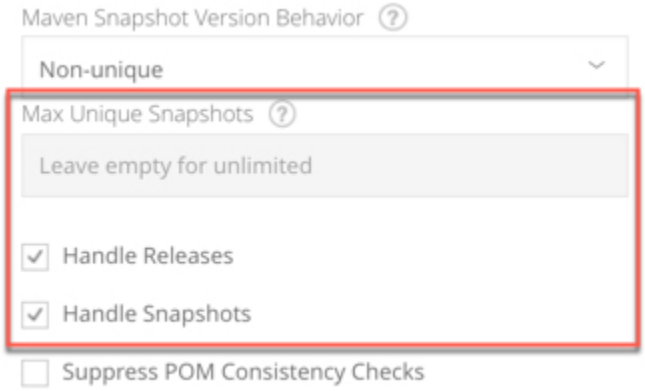
Setting Priority for Safe Remote and Local Repositories

From Artifactory version 7.16.3, this feature is currently supported for Docker, PyPI, RubyGems, and NPM packages.

You can declare local and remote repositories as 'safe' by enabling the 'Priority Resolution' field for local and remote repositories. 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.

Local and Remote Repositories

In addition to the settings above, Local and Remote repositories share the following settings in the type-specific section for relevant package types.



Max Unique Snapshots	Specifies the maximum number of unique snapshots of the same artifact that should be stored. Once this number is reached and a new snapshot is uploaded, the oldest stored snapshot is removed automatically. Blank (default) indicates that there is no limit on the number of unique snapshots.
Handle Releases	If set, Artifactory allows you to deploy release artifacts into this repository.
Handle Snapshots	If set, Artifactory allows you to deploy snapshot artifacts into this repository.