npm Registry

Overview

Artifactory provides full support for managing npm packages and ensures optimal and reliable access to *npmjs.org*. Aggregating multiple npm registries under a virtual repository Artifactory provides access to all your npm packages through a single URL for both upload and download.

As a fully-fledged npm registry on top of its capabilities for advanced artifact management, Artifactory's support for npm provides:

1. The ability to provision npm packages from Artifactory to the npm command line tool from all repository types.
2. Calculation of Metadata for npm packages hosted in Artifactory's local repositories.
3. Access to remote npm registries (such as `https://registry.npmjs.org`) through Remote Repositories which provide the usual proxy and caching functionality.
4. The ability to access multiple npm registries from a single URL by aggregating them under a Virtual Repository. This overcomes the limitation of the npm client which can only access a single registry at a time.
5. Compatibility with the npm command line tool to deploy and remove packages and more.
6. Support for flexible npm repository layouts that allow you to organize your npm packages and assign access privileges according to projects or development teams.
7. Support for validating remote npm repository metadata.

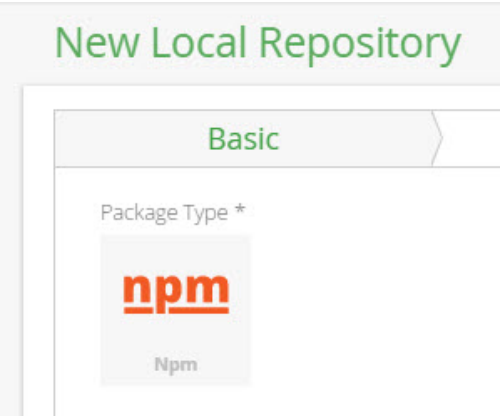
Npm version support

Artifactory supports npm version 1.4.3 and above.

Configuration

Local npm Registry

To enable calculation of npm package metadata in local repositories, set the **Package Type** to **npm** when you create the repository:



Page Contents

Integration Benefits

JFrog Artifactory and npm Registires

Repository Layout

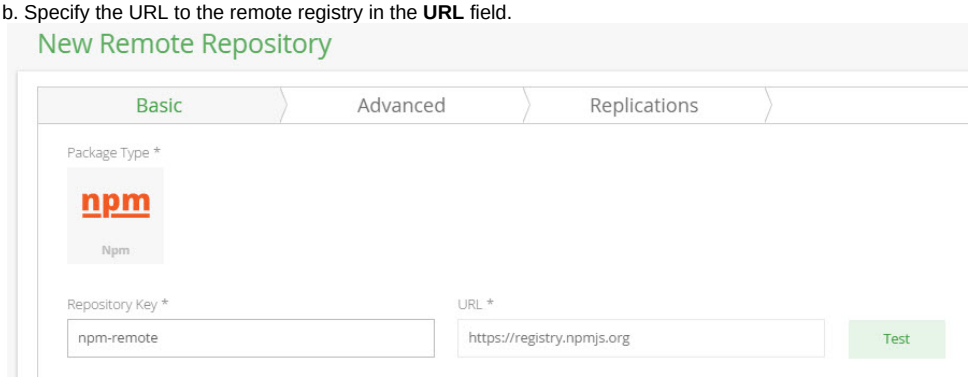
Artifactory allows you to define any layout for your npm registries. In order to upload packages according to your custom layout, you need to package your npm files using the `npm pack` command. This creates the .tgz file for your package which you can then upload to any path within your local npm repository.

Remote npm Registry

A Remote Repository defined in Artifactory serves as a caching proxy for a registry managed at a remote URL such as `https://registry.npmjs.org`. Artifacts (such as TGZ files) requested from a remote repository are cached on demand. You can remove downloaded artifacts from the remote repository cache, however, you can not manually deploy artifacts to a remote npm registry.

To define a remote repository to proxy a remote npm registry:

1. In the **Admin** module, under **Repositories | Remote,** click **New**.
2. In the **New Remote Repository** dialog:
   a. Set the **Package Type** to **npm** and the **Repository Key** value.
   b. Specify the URL to the remote registry in the **URL** field.



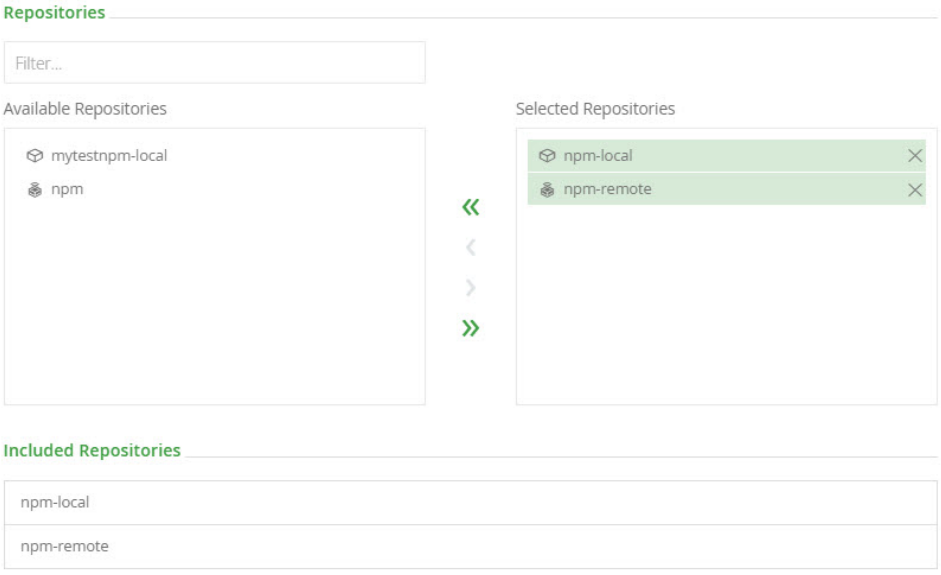
3. Click **Save & FInish**.

Virtual npm Registry

A Virtual Repository defined in Artifactory aggregates packages from both local and remote repositories.
This allows you to access both locally hosted npm packages and remote proxied npm registries from a single URL defined for the virtual repository.

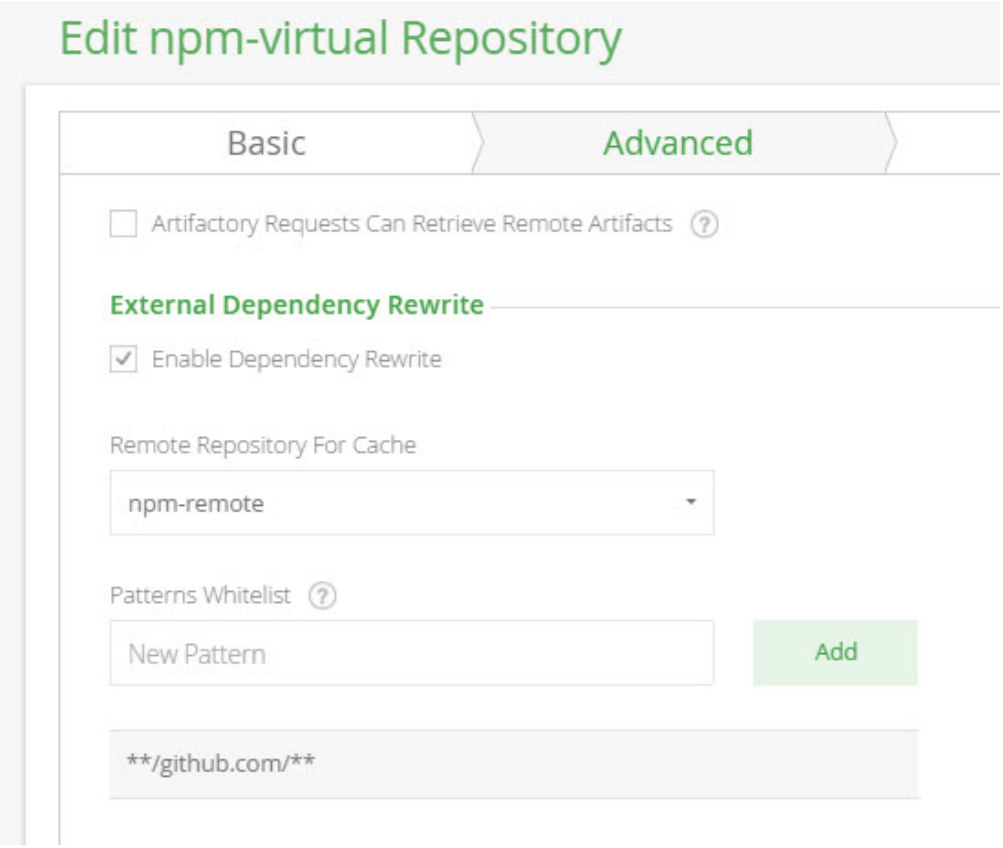
To define a virtual npm registry:

1. Create a virtual repository, set the **Package Type** to **npm.**
2. Select the underlying local and remote npm registries to include in the **Basic** settings tab.



3. Click **Save & Finish** to create the repository.

Advanced Configuration



The fields under **External Dependency Rewrite** are connected to automatically rewriting external dependencies for npm packages that require them.

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 Whitelist	A whitelist 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 *`github.com`*, you should add *`**github.com**`* (and remove the default **** expression).

Using the npm Command Line

Npm repositories must be prefixed with api/npm in the path

When accessing an npm repository through Artifactory, the repository URL must be prefixed with **api/npm** in the path. This applies to all npm commands including `npm install` and `npm publish`.

For example, if you are using Artifactory standalone or as a local service, you would access your npm repositories using the following URL:

- *`http://localhost:8081/artifactory/`**`api/npm`**`/<repository key>`*

Or, if you are using Artifactory SaaS, the URL would be:

- *`https://<server name>.jfrog.io/<server name>/`**`api/npm`**`/<repository key>`*

To use the npm command line you need to make sure npm is installed. Npm is included as an integral part of recent versions of Node.js.

Please refer to Installing Node.js via package manager on GitHub or the npm README page.

Once you have created your npm repository, you can select it in the Tree Browser and click **Set Me Up** to get code snippets you can use to change your npm registry URL, deploy and resolve packages using the npm command line tool.

Setting the Default Registry

For your npm command line client to work with Artifactory, you first need to set the default npm registry with an Artifactory npm repository using the following command (the example below uses a repository called *npm-repo*):

Replacing the default registry

```
npm config set registry http://<ARTIFACTORY_SERVER_DOMAIN>:8081/artifactory/api/npm/npm-repo/
```

For scoped packages, use the following command:

```
npm config set @<SCOPE>:registry http://<ARTIFACTORY_SERVER_DOMAIN>:8081/artifactory/api/npm/npm-repo/
```

We recommend referencing a Virtual Repository URL as a registry. This gives you the flexibility to reconfigure and aggregate other external sources and local repositories of npm packages you deployed.

Note that if you do this, you need to use the *--registry* parameter to specify the local repository into which you are publishing your package when using the `npm publish` command.

Authenticating the npm Client

Once you have set the default registry, you need to authenticate the npm client to Artifactory in one of two ways:

- Running the `npm login` command
- Using basic authentication.

Using npm login

Authentication using `npm login` was introduced in version 5.4.

Run the following command in your npm client. When prompted, provide your Artifactory login credentials:

```
npm login
```

Upon running this command, Artifactory creates a non-expirable access token which the client uses for authentication against Artifactory for subsequent and `npm installnpm publish` actions.

If the token is removed from Artifactory, the client will have to log in again to receive a new token.

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`)
- Set `always-auth = true`

.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, API Key or Access Token

Here is an example of the response:

```
_auth = YWRtaW46e0RFU2VkZX1uOFRaaXh1Y0t3bHN4c2RCTVIwNjF3PT0=
email = myemail@email.com
always-auth = true
```

If, in addition, you are also working with scoped packages, you also need to run the following command:

```
curl -uadmin:<CREDENTIAL> http://<ARTIFACTORY_SERVER_DOMAIN>:8081/artifactory/api/npm/npm-repo/auth/<SCOPE>
```

Where <CREDENTIAL> is your Artifactory password or API Key.

Paste the response to this command in the `~/.npmrc` file on your machine (in Windows, `%USERPROFILE%/.npmrc`).

Resolving npm Packages

Once the npm command line tool is configured, every *`npm install`* command will fetch packages from the npm repository specified above.
For example:

```
$ npm install request
npm http GET http://localhost:8081/artifactory/api/npm/npm-repo/request
npm http 200 http://localhost:8081/artifactory/api/npm/npm-repo/request
npm http GET http://localhost:8081/artifactory/api/npm/npm-repo/request/-/request-2.33.0.tgz
npm http 200 http://localhost:8081/artifactory/api/npm/npm-repo/request/-/request-2.33.0.tgz
```

---

npm 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
`npm audit`	Returns a vulnerability report based on the dependency tree sent by the npm client that is generated by https://npmjs.com/ and optionally enhanced by Jfrog Xray.
`npm audit fix`	Fetches the same report as `npm audit` and attempts to automatically act upon the recommendations in the report.

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.

Learn more about npm audit.

npm Publish (Deploying Packages)

Setting Your Credentials

The npm command line tool requires that sensitive operations, such as `publish` are authenticated as described under Authenticating the npm Client a bove.

Deploying Your Packages

There are two ways to deploy packages to a local repository:

- Edit your *package.json* file and add a publishConfig section to a local repository:
  *"publishConfig":{"registry":"http://localhost:8081/artifactory/**api/npm**/npm-local/"}*
- Provide a local repository to the `npm publish` command:
  *npm publish --registry http://localhost:8081/artifactory/**api/npm**/npm-local/*

Specifying the Latest Version

By default, the "latest" version of a package in an npm registry in Artifactory is the one with the highest SemVer version number. You can override this so that the most recently uploaded package is returned by Artifactory as the "latest" version. To do so, in Artifactory's system.properties file, add or set:

```
artifactory.npm.tag.tagLatestByPublish = true
```

Working with Artifactory without Anonymous Access

By default, Artifactory allows anonymous access to npm repositories. This is defined in the **Admin** module under **Security | General**. For details, please refer to Allow Anonymous Access.
If you want to trace how users interact with your repositories you need to disable the Allow Anonymous Access setting. This means that users will be required to enter their username and password as described in Setting Your Credentials above.

Using OAuth Credentials

Artifactory uses GitHub Enterprise as its default OAuth provider. If you have an account, you may use your GitHub Enterprise login details to be authenticated when using the `npm login` command.

npm Search

Artifactory supports a variety of ways to search artifacts. For details, please refer to Searching Artifacts.
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.

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.

Cleaning 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*.

---

npm Scope Packages

Artifactory fully supports npm scope packages. The support is transparent to the user and does not require any different usage of the npm client.

Npm 'slash' character encoding

By default, the npm client encodes slash characters ('/') to their ASCII representation ("%2f") before communicating with the npm registry. If you are running Tomcat as your HTTP container (the default for Artifactory), this generates an "HTTP 400" error since Tomcat does not allow encoded slashes by default. In order to work with npm scoped packages, you can override this default behavior by defining the following property in the *catalina.properties* file of your Tomcat:

```
org.apache.tomcat.util.buf.UDecoder.ALLOW_ENCODED_SLASH=true
```

You can also add **-Dorg.apache.tomcat.util.buf.UDecoder.ALLOW_ENCODED_SLASH=true** to the $ARTIFACTORY_HOME/etc/default for service installations or $ARTIFACTORY_HOME/bin/artifactory.default file. Note that since Artifactory **version 4.4.3**, the bundled Tomcat is configured by default to enable encoded slashes. If you are using a previous version you will need to adjust the Tomcat property above.

URL decoding and reverse proxy

If Artifactory is running behind a reverse proxy, make sure to disable URL decoding on the proxy itself in order to work with npm scope packages.

For Apache, add the "*AllowEncodedSlashes NoDecode*" directive inside the <VirtualHost *:xxx> block.

Configuring the npm Client for a Scope Registry

Using Login Credentials

Scopes can be associated with a separate registry. This allows you to seamlessly use a mix of packages from the public npm registry and one or more private registries.

For example, you can associate the scope *@jfrog* with the registry *http://localhost:8081/artifactory/api/npm/npm-local/* by manually altering your *~/.npmrc* file and adding the following configuration:

```
@jfrog:registry=http://localhost:8081/artifactory/api/npm/npm-local/
//localhost:8081/artifactory/api/npm/npm-local/:_password=cGFzc3dvcmQ=
//localhost:8081/artifactory/api/npm/npm-local/:username=admin
//localhost:8081/artifactory/api/npm/npm-local/:email=myemail@email.com
//localhost:8081/artifactory/api/npm/npm-local/:always-auth=true
```

Getting .npmrc entries directly from Artifactory

From Artifactory 3.5.3, you can use the following command to get these strings directly from Artifactory:

```
$ curl -uadmin:password "http://localhost:8081/artifactory/api/npm/npm-local/auth/jfrog"
@jfrog:registry=http://localhost:8081/artifactory/api/npm/npm-local/
//localhost:8081/artifactory/api/npm/npm-local/:_password=QVA1N05OaHZTMnM5Qk02RkR5RjNBVmF4TVFl
//localhost:8081/artifactory/api/npm/npm-local/:username=admin
//localhost:8081/artifactory/api/npm/npm-local/:email=admin@jfrog.com
//localhost:8081/artifactory/api/npm/npm-local/:always-auth=true
```

User email is required

When using scope authentication, npm expects a valid email address. Please make sure you have included your email address in your Artifactory user profile.

The password is just a base64 encoding of your Artifactory password, the same way used by the old authentication configuration.

Recommend npm command line tool version 2.1.9 and later.

While npm scope packages have been available since version 2.0 of the npm command line tool, we highly recommend using npm scope packages with Artifactory only from version 2.1.9 of the npm command line tool.

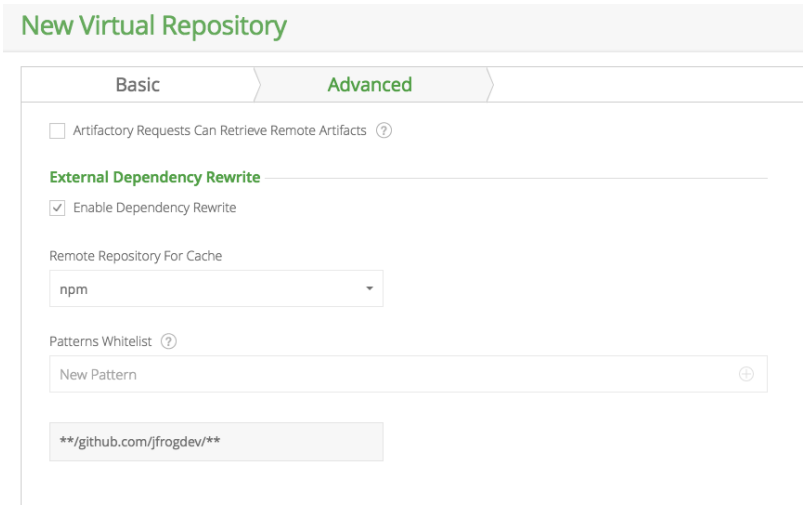
Automatically Rewriting External Dependencies

Packages requested by the npm client frequently use external dependencies as defined in the packages' *package.json* file. These dependencies may, in turn, need additional dependencies. Therefore, when downloading an npm package, you may not have full visibility into the full set of dependencies that your original package needs (whether directly or transitively). As a result, you are at risk of downloading malicious dependencies from unknown external resources.

To manage this risk, and maintain the best practice of consuming external packages through Artifactory, you may specify a "safe" whitelist from which dependencies may be downloaded, cached in Artifactory and configure to rewrite the dependencies so that the npm client accesses dependencies through a virtual repository as follows:

- Select the **Enable Dependency Rewrite** checkbox in the npm virtual repository advanced configuration.
- Specify a whitelist 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
```

Rewriting 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 whitelist,
    a. Artifactory will download the dependency from the external resource.
    b. Artifactory will cache the dependency in the remote repository configured to cache the external dependency.
    c. Artifactory will then modify the dependency's entry in the package's *package.json* file indicating its new location in the Artifactory remote repository cache before returning it to the Npm client.
3. Consequently, every time the npm client needs to access the dependency, it will be provisioned from its new location in the Artifactory remote repository cache.

SemVer Support

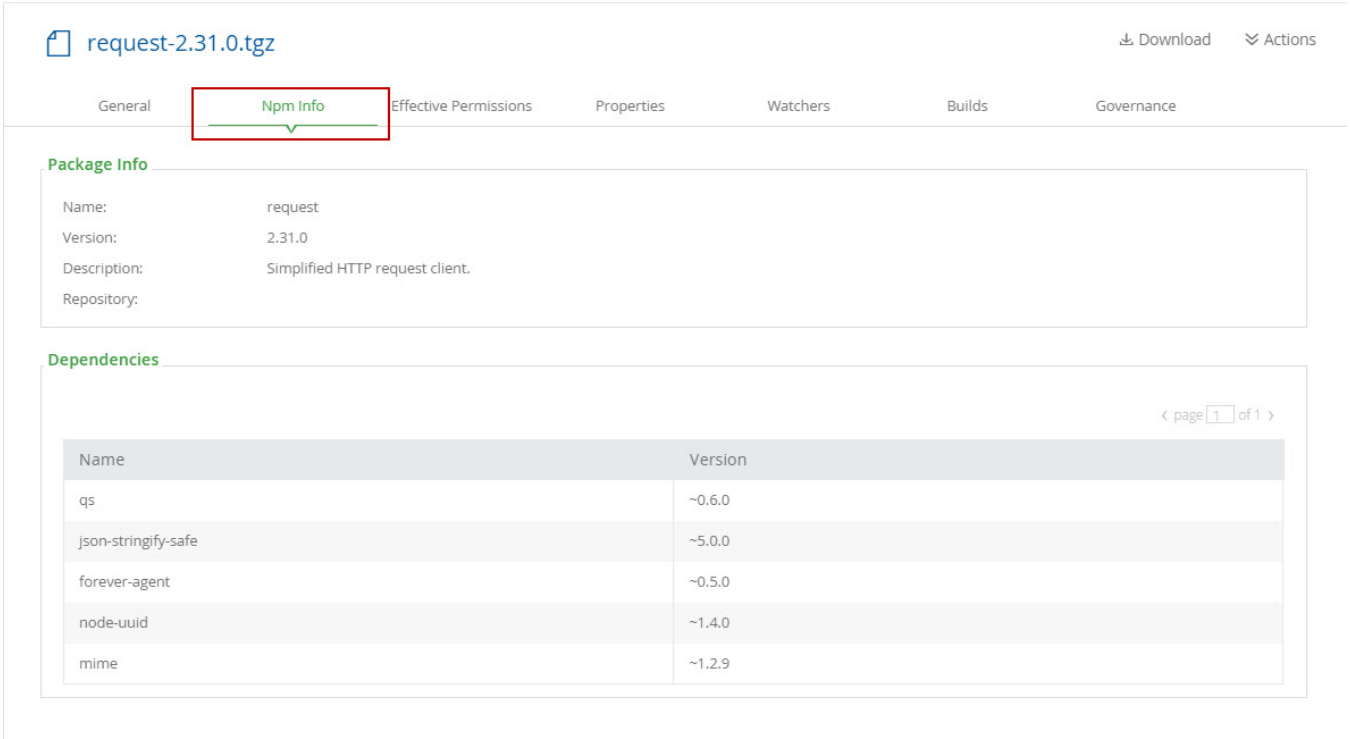
From Artifactory 6.17, the external dependency rewrite feature for the npm virtual repository supports additional SemVer expressions, such as semver:4.x.0.

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.

Viewing Individual npm Package Information

Artifactory lets you view selected metadata of an npm package directly from the UI.

In the **Tree Browser,** drill down to select the TGZ file you want to inspect. The metadata is displayed in the n**pm Info** tab.



---

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.

Watch the Screencast