plugin-installation-manager-tool

Public

1 Branch 72 Tags Go to file t Go to file Add file Code About

renovate[bot] Update dependency org.wiremock:wiremock to v3.9.2 ✓

187024e · 5 days ago

.github	Update softprops/action-gh-rel...	2 weeks ago
.mvn	Update dependency io.jenkins.t...	6 months ago
plugin-management-cli	[maven-release-plugin] prepare ...	3 weeks ago
plugin-management-libr...	Update dependency org.wiremo...	5 days ago
.gitattributes	Configure explicit line terminati...	last year
.gitignore	Add a custom user agent (#314)	3 years ago
CONTRIBUTING.md	Add a contributing guide (#536)	last year
Jenkinsfile	Update Jenkinsfile	4 months ago
LICENSE	Add MIT License	5 years ago
README.md	Updated default war file locatio...	last year
pom.xml	Update dependency net.bytebu...	5 days ago

Plugin Manager CLI tool for Jenkins

#cli #jenkins #plugin-manager

#hacktoberfest

Readme

MIT license

Code of conduct

Security policy

Activity

Custom properties

394 stars

20 watching

202 forks

Report repository

Releases 60

2.13.2 Latest

3 weeks ago

+ 59 releases

README Code of conduct MIT license Security

Plugin Installation Manager Tool for Jenkins

changelog v2.13.2 downloads 1.4M chat on gitter

The plugin manager downloads plugins and their dependencies into a folder so that they can be easily imported into an instance of Jenkins. The goal of this tool is to replace the Docker install-plugins.sh script and the many other implementations of plugin management that have been recreated across Jenkins. The tool also allows users to see more information about the plugins they are downloading, such as available updates and security warnings. By default, plugins will be downloaded; the user can specify not to download plugins using the --no-download option.

+ 42 contributors

Languages

Java 99.7% Other 0.3%

Usage

- Getting Started
- CLI Options
- Advanced configuration
- Plugin Input Format
- Updating plugins
- Examples
- Proxy Support
- Other Information

Getting Started

Download the latest jenkins-plugin-manager jar from here and run it as shown below.

```
java -jar jenkins-plugin-manager-*.jar --war /your/path/to/jenkins.war --plugin-download-directory /your/path/to/plugins/ --plu
```

Alternatively, build and run the plugin manager yourself from source:

```
mvn clean install
java -jar plugin-management-cli/target/jenkins-plugin-manager-*.jar --war /file/path/jenkins.war --plugin-download-directory /y
```

If you use a Jenkins docker image the plugin manager can be invoked inside the running container via the bundled `jenkins-plugin-cli` shell script:

```
docker cp /your/path/to/plugins.txt <container_name>:/tmp/plugins.txt
docker exec -it <container_name> /bin/bash
jenkins-plugin-cli --plugin-file /tmp/plugins.txt --plugins delivery-pipeline-plugin:1.3.2 deployit-plugin
cp -r -p /usr/share/jenkins/ref/plugins/. /var/jenkins_home/plugins/.
exit
```

CLI Options

- `--plugin-file` or `-f` : (optional) Path to the plugins.txt, or plugins.yaml file, which contains a list of plugins to install. If this file does not exist, or if the file exists, but does not have a .txt or .yaml/.yml extension, then an error will be thrown.
- `--plugin-download-directory` or `-d` : (optional) Directory in which to install plugins. This configuration can also be made via the PLUGIN_DIR environment variable. The directory will be first deleted, then recreated. If no directory configuration is provided, the defaults are C:\ProgramData\Jenkins\Reference\Plugins if the detected operating system is Microsoft Windows, or /usr/share/jenkins/ref/plugins otherwise.
- `--plugins` or `-p` : (optional) List of plugins to install (see plugin format below), separated by a space.
- `--clean-download-directory` : (optional) If sets, cleans the plugin download directory before plugin installation. Otherwise the tool performs plugin download and reports compatibility issues, if any.
- `--jenkins-version` : (optional) Version of Jenkins to be used. If not specified, the plugin manager will try to extract it from the WAR file or other sources. The argument can be also set using the `JENKINS_VERSION` environment variable.
- `--war` or `-w` : (optional) Path to Jenkins war file. If no war file is entered, it will default to a specific location based on user's OS. In case of Windows, it will default to C:\ProgramData\Jenkins\jenkins.war and in case of OS other than Windows, it will default to /usr/share/java/jenkins.war. Plugins that are already included in the Jenkins war will only be downloaded if their required version is newer than the one included.
- `--list` or `-l` : (optional) Lists plugin names and versions of: installed plugins (plugins that already exist in the plugin directory), bundled plugins (non-detached plugins that exist in the war file), plugins that will be downloaded (highest required versions of the requested plugins and dependencies that are not already installed), and the effective plugin set (the highest versions of all plugins that are already installed or will be installed)
- `--verbose` : (optional) Show additional information about plugin dependencies and the download process
- `--hide-security-warnings` : (optional) Hide if any of the user specified plugins have security warnings
- `--view-all-security-warnings` : (optional) Show all plugins that have security warnings.
- `--available-updates` : (optional) Show if any requested plugins have newer versions available. If a Jenkins version-specific update center is available, the latest plugin version will be determined based on that update center's data.
- `--output {stdout,yaml,txt}` : (optional) Format to output plugin updates file in, stdout is the default.
- `--latest {true,false}` : (optional) Set to false to download the minimum required version of all dependencies.
- `--latest-specified` : (optional) (advanced) Download latest dependencies of any plugin that is requested to have the latest version. All other plugin dependency versions are determined by the update center metadata or the plugin MANIFEST.MF.
- `--jenkins-update-center` : (optional) Sets the main update center filename, which can also be set via the JENKINS_UC environment variable. If a CLI option is entered, it will override what is set in the environment variable. If not set via CLI option or environment variable, will default to https://updates.jenkins.io/update-center.actual.json
- `--jenkins-experimental-update-center` : (optional) Sets the experimental update center, which can also be set via the JENKINS_UC_EXPERIMENTAL environment variable. If a CLI option is entered, it will override what is set in the environment variable. If not set via CLI option or environment variable, will default to https://updates.jenkins.io/experimental/update-center.actual.json
- `--jenkins-incrementals-repo-mirror` : (optional) Sets the incrementals repository mirror, which can also be set via the JENKINS_INCREMENTALS_REPO_MIRROR environment variable. If a CLI option is entered, it will override what is set in the environment variable. If not set via CLI option or environment variable, will default to https://repo.jenkins-ci.org/incrementals.
- `--jenkins-plugin-info` : (optional) Sets the location of plugin information, which can also be set via the JENKINS_PLUGIN_INFO environment variable. If a CLI option is provided, it will override what is set in the environment variable. If not set via CLI option or environment variable, will default to https://updates.jenkins.io/current/plugin-versions.json.
- `--version` or `-v` : (optional) Displays the plugin management tool version and exits.

- `--no-download` : (optional) Do not download plugins. By default plugins will be downloaded.
- `--skip-failed-plugins` : (optional) Adds the option to skip plugins that fail to download - CAUTION should be used when passing this flag as it could leave Jenkins in a broken state.
- `--credentials` : (optional) Comma-separated list of credentials to use for Basic Authentication for specific hosts (and optionally ports). Each value must adhere to format `<host>[:port]:<username>:<password>` . The password must not contain a `,` ! The credentials are not used preemptively.

Advanced configuration

- `CACHE_DIR` : used to configure the directory where the plugins update center cache is located. By default it will be in `~/.cache/jenkins-plugin-management-cli` , if the user doesn't have a home directory when it will go to: `$(pwd)/.cache/jenkins-plugin-management-cli` .
- `JENKINS_UC_DOWNLOAD` : *DEPRECATED* use `JENKINS_UC_DOWNLOAD_URL` instead.
- `JENKINS_UC_DOWNLOAD_URL` : used to configure a custom URL from where plugins will be downloaded from. When this value is set, it replaces the plugin download URL found in the `update-center.json` file with `${JENKINS_UC_DOWNLOAD_URL}` . Often used to cache or to proxy the Jenkins plugin download site. If set then all plugins will be downloaded through that URL.
- `JENKINS_UC_HASH_FUNCTION` : used to configure the hash function which checks content from UCs. Currently `SHA1` (deprecated), `SHA256` (default), and `SHA512` can be specified.

Plugin Input Format

The expected format for plugins in the .txt file or entered through the `--plugins` CLI option is `artifact ID:version` or `artifact ID:url` or `artifact:version:url`

Use plugin artifact ID, without -plugin extension. If a plugin cannot be downloaded, -plugin will be appended to the name and download will be retried. This is for cases in which plugins don't follow the rules about artifact ID (i.e. docker plugin).

The version and download url are optional. By default, the latest version of the plugin will be downloaded. If both a version and a url are supplied, the version will not be used to determine the plugin download location and the library will attempt to download the plugin from the given url.

The following custom version specifiers can also be used:

- `latest` - downloads the latest version from a version specific update center if one exists for the version in the Jenkins war file. If no version specific update center exists, will use the main update center https://updates.jenkins.io
- `experimental` - downloads the latest version from the experimental update center, which offers Alpha and Beta versions of plugins. Default value: https://updates.jenkins.io/experimental
- `incrementals;org.jenkins-ci.plugins.workflow;2.19-rc289.d09828a05a74` - downloads the plugin from the incrementals repo. For this option you need to specify groupId of the plugin. Note that this value may change between plugin versions without notice. More information on incrementals and their use for Docker images can be found here.

A set of plugins can also be provided through a YAML file, using the following format:

```yaml
plugins:
  - artifactId: git
    source:
      version: latest
  - artifactId: job-import-plugin
    source:
      version: 2.1
  - artifactId: docker
  - artifactId: cloudbees-bitbucket-branch-source
    source:
      version: 2.4.4
  - artifactId: script-security
    source:
      url: http://ftp-chi.osuosl.org/pub/jenkins/plugins/script-security/1.56/script-security.hpi
  - artifactId: workflow-step-api
    groupId: org.jenkins-ci.plugins.workflow
    source:
      version: 2.19-rc289.d09828a05a74
  ...
```

As with the plugins.txt file, version and URL are optional. If no version is provided, the latest version is used by default. If a groupId is provided, the tool will try to download the plugin from the Jenkins incrementals repository.

Updating plugins

The CLI can output a new file with a list of updated plugin references.

Text format:

```
$ java -jar jenkins-plugin-manager-*.jar --available-updates --output txt --plugins mailer:1.31
```

Result:

```
mailer:1.32
```

YAML format:

```
$ java -jar jenkins-plugin-manager-*.jar --available-updates --output yaml --plugins mailer:1.31
```

Result:

```
plugins:
- artifactId: "mailer"
  source:
    version: "1.32"
```

Human readable:

```
$ java -jar jenkins-plugin-manager-*.jar --available-updates --plugins mailer:1.31
```

Result:

```
Available updates:
mailer (1.31) has an available update: 1.32
```

Examples

If a URL is included, then a placeholder should be included for the version. Examples of plugin inputs:

- `github-branch-source` - will download the latest version
- `github-branch-source:latest` - will download the latest version
- `github-branch-source:2.5.3` - will download version 2.5.3
- `github-branch-source:experimental` - will download the latest version from the experimental update center
- `github-branch-source:2.5.2:https://updates.jenkins.io/2.121/latest/github-branch-source.hpi` - will download version of plugin at url regardless of requested version
- `github-branch-source:https://updates.jenkins.io/2.121/latest/github-branch-source.hpi` - will treat the url like the version, which is not likely the behavior you want
- `github-branch-source::https://updates.jenkins.io/2.121/latest/github-branch-source.hpi` - will download plugin from url

If a plugin to be downloaded from the incrementals repository is requested using the -plugins option from the CLI, the plugin name should be enclosed in quotes, since the semi-colon is otherwise interpreted as the end of the command.

```
java -jar jenkins-plugin-manager-*.jar -p "workflow-support:incrementals;org.jenkins-ci.plugins.workflow;2.19-rc289.d09828a05a74"
```

Proxy Support

Proxy support is available using standard Java networking system properties `http.proxyHost` and `http.proxyPort`. Note that this provides only basic NTLM support and you may need to use an authentication proxy like CNTLM to cover more advanced authentication use cases.

```
# Example using proxy system properties
java -Dhttp.proxyPort=3128 -Dhttp.proxyHost=myproxy.example.com -Dhttps.proxyPort=3128 -Dhttps.proxyHost=myproxy.example.com -j
```

Other Information

The plugin manager tries to use update center data to get the latest information about a plugin's dependencies. If this information is unavailable, it will use the dependency information from the downloaded plugin's MANIFEST.MF file. By default, the versions of the plugin dependencies are determined by the update center metadata or the plugin MANIFEST.MF file, but the user can specify other behavior using the `latest` or `latest-specified` options.

For plugins listed in a .txt file, each plugin must be listed on a new line. Comments beginning with `#` will be filtered out.

Support for downloading plugins from maven is not currently supported. JENKINS-58217

When using `--latest` you may run into a scenario where the jenkins update mirror contains the directory of the newer version of a plugin(release in progress), regardless of if there is a jpi to download, which results in a download failure. It's recommended that you pin your plugin requirement versions until the mirror has been updated to more accurately represent what is available. More information on this challenge can be found here, and here.

The version-pinning behavior of this plugin installer has changed compared to the previous Jenkins plugin installer. By default, `--latest` option defaults to `true`, which means that even if you pass a list of pinned versions, these may fail to be installed correctly if they or some other dependency has a newer latest version available. In order to use *only* pinned versions of plugins, you must pass `--latest=false`. **NOTE:** When a new dependency is added to a plugin, it won't get updated until you notice that it's missing from your plugin list. (Details here: #250)

Contributions

Thanks to all our contributors! Check out our CONTRIBUTING file to learn how to get started with issues.

- Feel free to pick a task from the Issues and get started.