Pipelines Extension Usage

Overview

This page covers essential features of using Pipelines extensions, including versioning.

Page Contents

Namespaces

Extension steps and resources are defined within a *namespace*, to ensure that all steps and resources in a set have unique names. Namespaces are specified through the repository path of the step or resource model definition.

- Extension step definitions must be stored in a subdirectory path of the form: `steps/<namespace>/<stepTypeName>`.
- Extension resource definitions must be stored in a subdirectory path of the form: `resources/<namespace>/<resourceTypeName>`.

> If no namespace is specified, Pipelines defaults to the `jfrog` namespace for standard Pipelines resources, or standard generic or native steps.

Example

The sample extensions in our tutorial are in a `tutorial` namespace, as defined in the repository's directory structure:

`jfrog-pipelines-extensions-sample`	`/steps`	`/`**`tutorials`**	`/HealthCheck`
	`/resources`	`/`**`tutorials`**	`/test`

You can define as many steps or resources under a namespace directory as you need. For example:

`my_pipelines_extensions`	`/steps`	`/`**`mycompanyname`**	`/StepTypeA`
			`/StepTypeB`
			`/StepTypeC`
	`/resources`	`/`**`mycompanyname`**	`/ResourceTypeX`
			`/ResourceTypeY`
			`/ResourceTypeZ`

When used in a Pipelines DSL file, the extension must be referenced with its namespace to be recognized. For example:

pipelines.yml

```
resources:
  - name: MyResource
    type: mycompanyname/ResourceTypeX        # <-- extension resource
    configuration:
    ...

pipelines:
  - name: MyPipeline
    configuration:
      environmentVariables:
        readOnly:
          my_env_var: "hello"
    steps:
      - name: MyStep_1
        type: mycompanyname/StepTypeA        # <-- extension step
        configuration:
        ...
      - name: MyStep_2
        type: mycompanyname/StepTypeB        # <-- extension step
        configuration:
        ...
```

Versioning

All Pipelines extension steps and resources may be individually versioned through Git tags in their source control repository. These versions can then be released for general use through the Pipelines extensions management UI.

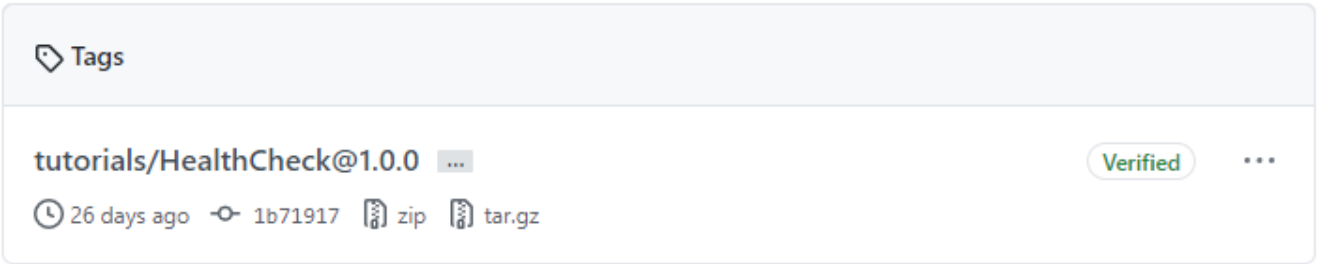
Tagging Extension Versions

A Git tag for versioning a step or resource must be of the form:

```
<namespace>/<TypeName>@<semver>
```

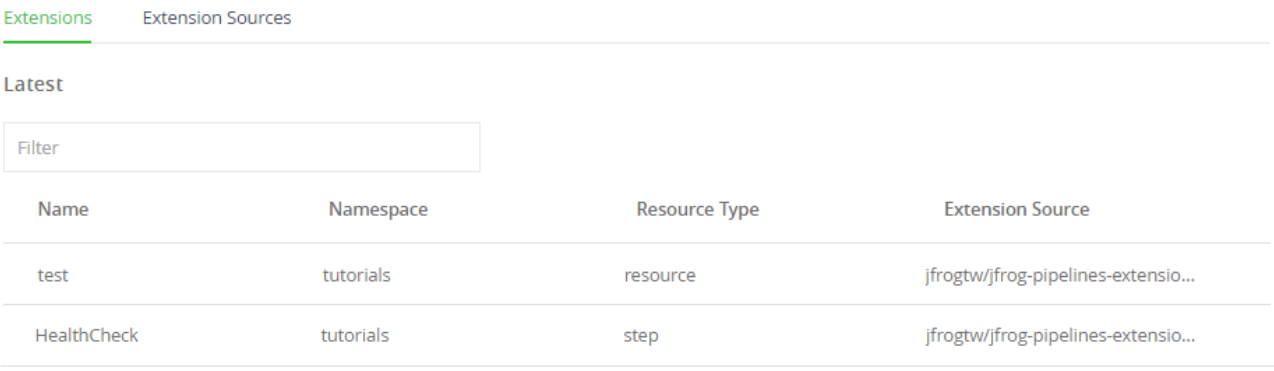
Where `<TypeName>` is the name of the extension step or resource type. The format of `<semver>` must be compliant with the semantic versioning standard (*Major.Minor.Patch*).

For example, to bind the currently committed version of the tutorial's `HealthCheck` step to version 1.0.0, you would draft a new release and assign it a tag `tutorials/HealthCheck@1.0.0`.

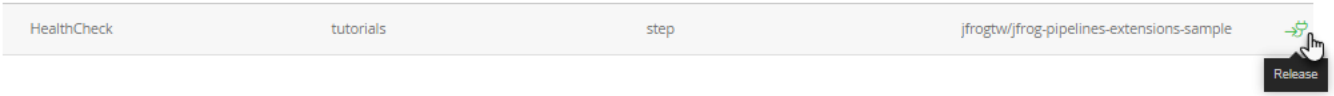


Releasing Extension Versions

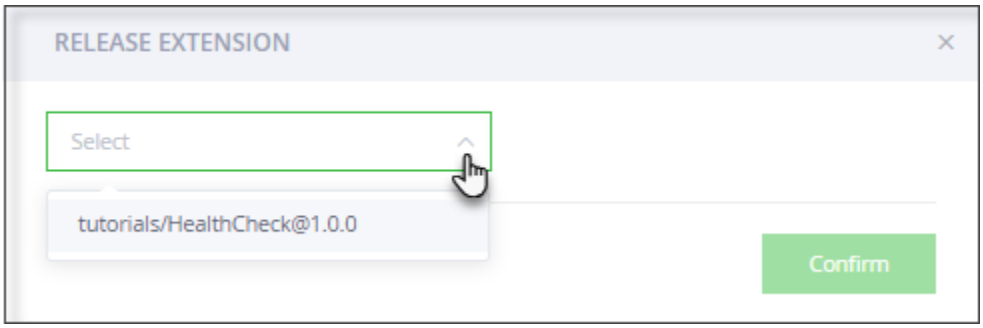
When the respository is added as an extension source, each extension loaded is the **Latest** version of that extension step or resource.



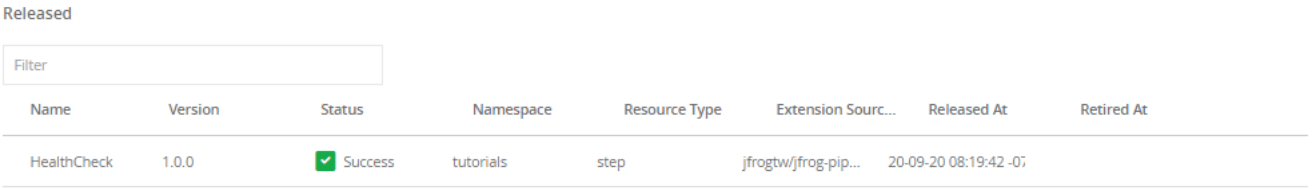
To release a properly Git tagged resource or step, hover over the rightmost region of its row to reveal the **Release** icon, then click it.



In the subsequent **Release Extension** dialog, select the release tag from the dropdown, then click **Confirm**.



Once confirmed, the version will be listed among the **Released** extensions.



The released version of the resource will now always be available to users by that version number, even afer a newer version is synced from the extension source.

Extensions can be retired from use on a specific date. Details of this procedure are described in Managing Pipelines Extensions.

Using Extension Versions

By default, a reference to an extension step or resource `type` always uses the latest synced version (listed in **Latest**).

To use a specific version (for example, one that is known to be good), you may qualify the `type` declaration to use a specific version by following it with the optional `syntaxVersion` tag.

For example, to use version 1.0.0 of the example `tutorial/HealthCheck` step:

Extension step with version

```
steps:
  - name: Step_1
    type: tutorials/HealthCheck
    syntaxVersion: 1.0.0           # Use a specific version
    configuration:
      ...
```

Similarly, you can use the `syntaxVersion` tag to qualify the `type` of a resource for a specific version.

Extension resource with version

```
resources:
  - name: MyResource
    type: tutorials/test
    syntaxVersion: 0.2.0           # Use a specific version
    configuration:
      ...
```

Version numbers that do not exist or that have been retired will result in an error when the pipeline is run.