Working with the RubyGems registry

In this article

You can configure RubyGems to publish a package to GitHub Packages and to use packages stored on GitHub Packages as dependencies in a Ruby project with Bundler.

GitHub Packages is available with GitHub Free, GitHub Pro, GitHub Free for organizations, GitHub Team, GitHub Enterprise Cloud, GitHub Enterprise Server 3.0 or higher, and GitHub AE.

GitHub Packages is not available for private repositories owned by accounts using legacy per-repository plans. Also, accounts using legacy per-repository plans cannot access registries that support granular permissions, because these accounts are billed by repository. For the list of registries that support granular permissions, see "About permissions for GitHub Packages." For more information, see "GitHub’s plans."

Prerequisites

- You must have RubyGems 2.4.1 or higher. To find your RubyGems version:

  ```
  gem --version
  ```

- You must have bundler 1.6.4 or higher. To find your Bundler version:

  ```
  $ bundle --version
  Bundler version 1.13.7
  ```

Authenticating to GitHub Packages

GitHub Packages only supports authentication using a personal access token (classic). For more information, see "Managing your personal access tokens."

You need an access token to publish, install, and delete private, internal, and public packages.

You can use a personal access token (classic) to authenticate to GitHub Packages or the GitHub API. When you create a personal access token (classic), you can assign the token different scopes depending on your needs. For more information about packages-related scopes for a personal access token (classic), see "About permissions for GitHub Packages."

To authenticate to a GitHub Packages registry within a GitHub Actions workflow, you can use:

- `GITHUB_TOKEN` to publish packages associated with the workflow repository.
- a personal access token (classic) with at least `read:packages` scope to install packages associated with other private repositories (which `GITHUB_TOKEN` can't access).

Authenticating in a GitHub Actions workflow

This registry supports granular permissions. For registries that support granular permissions, if your GitHub Actions workflow is using a personal access token to authenticate to a registry, we highly recommend you update your workflow to use the `GITHUB_TOKEN`. For guidance on updating your workflows that authenticate to a registry with a personal access token, see "Publishing and installing a package with GitHub Actions."

Note: The ability for GitHub Actions workflows to delete and restore packages using the REST API is currently in public beta and subject to change.

You can use a `GITHUB_TOKEN` in a GitHub Actions workflow to delete or restore a package using the REST API, if the token has `admin` permission to the package. Repositories that publish packages using a workflow, and repositories that you have explicitly connected to packages, are automatically granted `admin` permission to packages in the repository.

For more information about the `GITHUB_TOKEN`, see "Automatic token authentication." For more information about the best practices when using a registry in actions, see "Security hardening for GitHub Actions."

You can also choose to give access permissions to packages independently for GitHub Codespaces and GitHub Actions. For more information, see "Configuring a package's access control and visibility" and "Configuring a package's access control and visibility."

Authenticating with a personal access token

You must use a personal access token (classic) with the appropriate scopes to publish and install packages in GitHub Packages. For more information, see "Introduction to GitHub Packages."

To publish and install gems, you can configure RubyGems or Bundler to authenticate to GitHub Packages using your personal access token.

To publish new gems, you need to authenticate to GitHub Packages with RubyGems by editing your *~/.gem/credentials* file to include your personal access token (classic). Create a new *~/.gem/credentials* file if this file doesn't exist.

For example, you would create or edit a *~/.gem/credentials* to include the following, replacing TOKEN with your personal access token.

```
---
:github: Bearer TOKEN
```

To install gems, you need to authenticate to GitHub Packages by updating your gem sources to include `https://USERNAME:TOKEN@rubygems.pkg.github.com/NAMESPACE/`. You must replace:

- `USERNAME` with your GitHub username.

- `TOKEN` with your personal access token (classic).
- `NAMESPACE` with the name of the personal account or organization to which the gem is scoped.

If you would like your package to be available globally, you can run the following command to add your registry as a source.

```
gem sources --add https://USERNAME:TOKEN@rubygems.pkg.github.com/NAMESPACE/
```

To authenticate with Bundler, configure Bundler to use your personal access token (classic), replacing USERNAME with your GitHub username, TOKEN with your personal access token, and NAMESPACE with the name of the personal account or organization to which the gem is scoped.

```
bundle config https://rubygems.pkg.github.com/NAMESPACE USERNAME:TOKEN
```

Publishing a package

When you first publish a package, the default visibility is private. To change the visibility or set access permissions, see "Configuring a package's access control and visibility." For more information on creating your gem, see "Make your own gem" in the RubyGems documentation.

Note: If you publish a package that is linked to a repository, the package automatically inherits the access permissions of the linked repository, and GitHub Actions workflows in the linked repository automatically get access to the package, unless your organization has disabled automatic inheritance of access permissions. For more information, see "Configuring a package's access control and visibility."

1. Authenticate to GitHub Packages. For more information, see "Authenticating to GitHub Packages."

2. Build the package from the *gemspec* to create the *.gem* package. Replace `GEM_NAME` with the name of your gem.

   ```
   gem build GEM_NAME.gemspec
   ```

3. Publish a package to GitHub Packages, replacing `NAMESPACE` with the name of the personal account or organization to which the package will be scoped and `GEM_NAME` with the name of your gem package.

   Note: The maximum uncompressed size of a gem's `metadata.gz` file must be less than 2 MB. Requests to push gems that exceed that limit will fail.

   ```
   $ gem push --key github \
   --host https://rubygems.pkg.github.com/NAMESPACE \
   GEM_NAME-0.0.1.gem
   ```

Connecting a package to a repository

The RubyGems registry stores packages within your organization or personal account, and allows you to associate packages with a repository. You can choose whether to inherit permissions from a repository, or set granular permissions independently of a repository.

You can ensure gems will be linked to a repository as soon as they are published by including the URL of the GitHub repository in the `github_repo` field in `gem.metadata`. You can link multiple gems to the same repository.

```
gem.metadata = { "github_repo" => "ssh://github.com/OWNER/REPOSITORY" }
```

For information on linking a published package with a repository, see "Connecting a repository to a package."

Installing a package

You can use gems from GitHub Packages much like you use gems from *rubygems.org*. You need to authenticate to GitHub Packages by adding your GitHub user or organization as a source in the *~/.gemrc* file or by using Bundler and editing your *Gemfile*.

1. Authenticate to GitHub Packages. For more information, see "Authenticating to GitHub Packages."

2. For Bundler, add your GitHub user or organization as a source in your *Gemfile* to fetch gems from this new source. For example, you can add a new `source` block to your *Gemfile* that uses GitHub Packages only for the packages you specify, replacing `GEM_NAME` with the package you want to install from GitHub Packages and `NAMESPACE` with the personal account or organization to which the gem you want to install is scoped.

```
source "https://rubygems.org"

gem "rails"

source "https://rubygems.pkg.github.com/NAMESPACE" do
  gem "GEM_NAME"
end
```

3. For Bundler versions earlier than 1.7.0, you need to add a new global `source`. For more information on using Bundler, see the bundler.io documentation.

```
source "https://rubygems.pkg.github.com/NAMESPACE"
source "https://rubygems.org"

gem "rails"
gem "GEM_NAME"
```

4. Install the package:

```
gem install GEM_NAME --version "0.1.1"
```

Further reading

- "Deleting and restoring a package"