# 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.

**Note:** This package type may not be available for your instance, because site administrators can enable or disable each supported package type. For more information, see "Configuring package ecosystem support for your enterprise."

## 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 Enterprise Server 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 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@REGISTRY_URL/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 that owns the repository containing the gem.
- `REGISTRY_URL` with the URL for your instance's Rubygems registry. If your instance has subdomain isolation enabled, use `rubygems.HOSTNAME`. If your instance has subdomain isolation disabled, use `HOSTNAME/_registry/rubygems`. In either case, replace HOSTNAME with the hostname of your GitHub Enterprise Server instance.

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@REGISTRY_URL/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 that owns the repository containing the gem. Replace `REGISTRY_URL` with the URL for your instance's RubyGems registry. If your instance has subdomain isolation enabled, use `rubygems.HOSTNAME`. If your instance has subdomain isolation disabled, use `HOSTNAME/_registry/rubygems`. In either case, replace HOSTNAME with the hostname of your GitHub Enterprise Server instance.

```
bundle config https://REGISTRY_URL/NAMESPACE USERNAME:TOKEN
```

## Publishing a package

By default, GitHub publishes the package to an existing repository with the same name as the package. For example, when you publish `GEM_NAME` to the `octo-org` organization, GitHub Packages publishes the gem to the `octo-org/GEM_NAME` repository. For more information on creating your gem, see "Make your own gem" in the RubyGems documentation.

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 that owns the repository containing your project and `GEM_NAME` with the name of your gem package. Replace `REGISTRY_URL` with the URL for your instance's Rubygems registry. If your instance has subdomain isolation enabled, use `rubygems.HOSTNAME`. If your instance has subdomain isolation disabled, use `HOSTNAME/_registry/rubygems`. In either case, replace `HOSTNAME` with the host name of your GitHub Enterprise Server instance.

   **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://REGISTRY_URL/NAMESPACE \
   GEM_NAME-0.0.1.gem
   ```

## Publishing multiple packages to the same repository

To publish multiple gems to the same repository, you can include the URL to the GitHub repository in the `github_repo` field in `gem.metadata`. If you include this field, GitHub matches the repository based on this value, instead of using the gem name. Replace HOSTNAME with the host name of your GitHub Enterprise Server instance.

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

## 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 that owns the repository containing the gem you want to install. Replace `REGISTRY_URL` with the URL for your instance's Rubygems registry. If your instance has subdomain isolation enabled, use

`rubygems.HOSTNAME`. If your instance has subdomain isolation disabled, use `HOSTNAME/_registry/rubygems`. In either case, replace `HOSTNAME` with the host name of your GitHub Enterprise Server instance.

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

gem "rails"

source "https://REGISTRY_URL/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://REGISTRY_URL/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"