Working with the NuGet registry

In this article

You can configure the `dotnet` command-line interface (CLI) to publish NuGet packages to GitHub Packages and to use packages stored on GitHub Packages as dependencies in a .NET project.

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."

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."

Use the following command to authenticate to GitHub Packages in a GitHub Actions workflow using the `GITHUB_TOKEN` instead of hardcoding a personal access token in a nuget.config file in the repository:

```
dotnet nuget add source --username USERNAME --password ${{ secrets.GITHUB_TOKEN
}} --store-password-in-clear-text --name github
"https://nuget.pkg.github.com/NAMESPACE/index.json"
```

Replace `NAMESPACE` with the name of the personal account or organization to which your packages are scoped.

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

> 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).

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 authenticate to GitHub Packages with the `dotnet` command-line interface (CLI), create a *nuget.config* file in your project directory specifying GitHub Packages as a source under `packageSources` for the `dotnet` CLI client.

You must replace:

- `USERNAME` with the name of your personal account on GitHub.
- `TOKEN` with your personal access token (classic).
- `NAMESPACE` with the name of the personal account or organization to which your packages are scoped.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <packageSources>
        <clear />
        <add key="github"
value="https://nuget.pkg.github.com/NAMESPACE/index.json" />
    </packageSources>
    <packageSourceCredentials>
        <github>
            <add key="Username" value="USERNAME" />
            <add key="ClearTextPassword" value="TOKEN" />
        </github>
    </packageSourceCredentials>
</configuration>
```

Publishing a package

You can publish a package to GitHub Packages by authenticating with a *nuget.config* file, or by using the `--api-key` command line option with your GitHub personal access token (classic).

The NuGet 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.

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

If you specify a `RepositoryURL` in your `nuget.config` file, the published package will automatically be connected to the specified repository. For more information, see "Working with the NuGet registry." For information on linking an already-published package to a repository, see "Connecting a repository to a package."

Publishing a package using a GitHub personal access token as your API key

If you don't already have a personal access token to use for your account on GitHub.com, see "Managing your personal access tokens."

1. Create a new project. Replace `PROJECT_NAME` with the name you'd like to give the project.

   ```
   dotnet new console --name PROJECT_NAME
   ```

2. Package the project.

```
dotnet pack --configuration Release
```

3. Publish the package using your personal access token as the API key. Replace `PROJECT_NAME` with the name of the project, `1.0.0` with the version number of the package, and `YOUR_GITHUB_PAT` with your personal access token.

```
dotnet nuget push "bin/Release/PROJECT_NAME.1.0.0.nupkg"  --api-key
YOUR_GITHUB_PAT --source "github"
```

After you publish a package, you can view the package on GitHub. For more information, see "Viewing packages."

Publishing a package using a *nuget.config* file

When publishing, if you are linking your package to a repository, the `OWNER` of the repository specified in your *.csproj* file must match the `NAMESPACE` that you use in your *nuget.config* authentication file. Specify or increment the version number in your *.csproj* file, then use the `dotnet pack` command to create a *.nuspec* file for that version. For more information on creating your package, see "Create and publish a package" in the Microsoft 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. Create a new project. Replace `PROJECT_NAME` with the name you'd like to give the project.

```
dotnet new console --name PROJECT_NAME
```

3. Add your project's specific information to your project's file, which ends in *.csproj*. Make sure to replace:

   - `1.0.0` with the version number of the package.
   - `OWNER` with the name of the personal account or organization that owns the repository to which you want to link your package.
   - `REPOSITORY` with the name of the repository to which you want to connect your package.

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp3.0</TargetFramework>
    <PackageId>PROJECT_NAME</PackageId>
    <Version>1.0.0</Version>
    <Authors>AUTHORS</Authors>
    <Company>COMPANY_NAME</Company>
    <PackageDescription>PACKAGE_DESCRIPTION</PackageDescription>
    <RepositoryUrl>https://github.com/OWNER/REPOSITORY</RepositoryUrl>
```

```
  </PropertyGroup>

</Project>
```

4. Package the project.

```
dotnet pack --configuration Release
```

5. Publish the package using the `key` you specified in the *nuget.config* file. Replace `PROJECT_NAME` with the name of the project, and replace `1.0.0` with the version number of the package.

```
dotnet nuget push "bin/Release/PROJECT_NAME.1.0.0.nupkg" --source "github"
```

After you publish a package, you can view the package on GitHub. For more information, see "Viewing packages."

Publishing multiple packages to the same repository

To connect multiple packages to the same repository, use the same GitHub repository URL in the `RepositoryURL` fields in all *.csproj* project files. GitHub matches the repository based on that field.

The following example publishes the projects MY_APP and MY_OTHER_APP to the same repository:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp3.0</TargetFramework>
    <PackageId>MY_APP</PackageId>
    <Version>1.0.0</Version>
    <Authors>Octocat</Authors>
    <Company>GitHub</Company>
    <PackageDescription>This package adds a singing Octocat!</PackageDescription>
    <RepositoryUrl>https://github.com/my-org/my-repo</RepositoryUrl>
  </PropertyGroup>

</Project>
```

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp3.0</TargetFramework>
    <PackageId>MY_OTHER_APP</PackageId>
    <Version>1.0.0</Version>
    <Authors>Octocat</Authors>
    <Company>GitHub</Company>
    <PackageDescription>This package adds a dancing Octocat!</PackageDescription>
    <RepositoryUrl>https://github.com/my-org/my-repo</RepositoryUrl>
  </PropertyGroup>

</Project>
```

Installing a package

Using packages from GitHub in your project is similar to using packages from *nuget.org*. Add your package dependencies to your *.csproj* file, specifying the package name and version. For more information on using a *.csproj* file in your project, see "Working with NuGet packages" in the Microsoft documentation.

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

2. To use a package, add `ItemGroup` and configure the `PackageReference` field in the *.csproj* project file. Replace the `PACKAGE_NAME` value in `Include="PACKAGE_NAME"` with your package dependency, and replace the `X.X.X` value in `Version="X.X.X"` with the version of the package you want to use:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp3.0</TargetFramework>
    <PackageId>My-app</PackageId>
    <Version>1.0.0</Version>
   <Authors>Octocat</Authors>
    <Company>GitHub</Company>
   <PackageDescription>This package adds an Octocat!</PackageDescription>
    <RepositoryUrl>https://github.com/OWNER/REPOSITORY</RepositoryUrl>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="PACKAGE_NAME" Version="X.X.X" />
  </ItemGroup>

</Project>
```

3. Install the packages with the `restore` command.

```
dotnet restore
```

Troubleshooting

If you're using a `GITHUB_TOKEN` to authenticate to a GitHub Packages registry within a GitHub Actions workflow, the token cannot access private repository-based packages in a different repository other than where the workflow is running in. To access packages associated with other repositories, instead generate a personal access token (classic) with the `read:packages` scope and pass this token in as a secret.

Further reading

- "Deleting and restoring a package"