Working with the Gradle registry

In this article

You can configure Gradle to publish packages to the GitHub Packages Gradle registry and to use packages stored on GitHub Packages as dependencies in a Java project.

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."

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).

For more information about `GITHUB_TOKEN` used in GitHub Actions workflows, see "Automatic token authentication." For more information about using `GITHUB_TOKEN` with Gradle, see "Publishing Java packages with Gradle."

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."

You can authenticate to GitHub Packages with Gradle using either Gradle Groovy or Kotlin DSL by editing your *build.gradle* file (Gradle Groovy) or *build.gradle.kts* file (Kotlin DSL) file to include your personal access token (classic). You can also configure Gradle Groovy and Kotlin DSL to recognize a single package or multiple packages in a repository.

Replace REGISTRY_URL with the URL for your instance's Maven registry. If your instance has subdomain isolation enabled, use `maven.HOSTNAME`. If your instance has subdomain isolation disabled, use `HOSTNAME/_registry/maven`. In either case, replace HOSTNAME with the host name of your GitHub Enterprise Server instance.

Replace USERNAME with your GitHub username, TOKEN with your personal access token (classic), REPOSITORY with the name of the repository containing the package you want to publish, and OWNER with the name of the personal account or organization on GitHub that owns the repository. Because uppercase letters aren't supported, you must use lowercase letters for the repository owner even if the GitHub user or organization name contains uppercase letters.

> **Note:** GitHub Packages supports `SNAPSHOT` versions of Apache Maven. To use the GitHub Packages repository for downloading `SNAPSHOT` artifacts, enable SNAPSHOTS in the POM of the consuming project or your *~/.m2/settings.xml* file. For an example, see "Working with the Apache Maven registry."

Example using Gradle Groovy for a single package in a repository

```
plugins {
    id("maven-publish")
}
publishing {
    repositories {
        maven {
            name = "GitHubPackages"
            url = uri("https://REGISTRY_URL/OWNER/REPOSITORY")
            credentials {
                username = project.findProperty("gpr.user") ?:
System.getenv("USERNAME")
                password = project.findProperty("gpr.key") ?:
System.getenv("TOKEN")
            }
        }
    }
    publications {
        gpr(MavenPublication) {
            from(components.java)
        }
    }
}
```

Example using Gradle Groovy for multiple packages in the same repository

```
plugins {
    id("maven-publish") apply false
}
subprojects {
    apply plugin: "maven-publish"
    publishing {
        repositories {
            maven {
                name = "GitHubPackages"
                url = uri("https://REGISTRY_URL/OWNER/REPOSITORY")
                credentials {
                    username = project.findProperty("gpr.user") ?:
```

```
System.getenv("USERNAME")
                    password = project.findProperty("gpr.key") ?:
System.getenv("TOKEN")
                }
            }
        }
        publications {
            gpr(MavenPublication) {
                from(components.java)
            }
        }
    }
}
```

Example using Kotlin DSL for a single package in the same repository

```
plugins {
    `maven-publish`
}
publishing {
    repositories {
        maven {
            name = "GitHubPackages"
            url = uri("https://REGISTRY_URL/OWNER/REPOSITORY")
            credentials {
                username = project.findProperty("gpr.user") as String? ?:
System.getenv("USERNAME")
                password = project.findProperty("gpr.key") as String? ?:
System.getenv("TOKEN")
            }
        }
    }
    publications {
        register<MavenPublication>("gpr") {
            from(components["java"])
        }
    }
}
```

Example using Kotlin DSL for multiple packages in the same repository

```
plugins {
    `maven-publish` apply false
}
subprojects {
    apply(plugin = "maven-publish")
    configure<PublishingExtension> {
        repositories {
            maven {
                name = "GitHubPackages"
                url = uri("https://REGISTRY_URL/OWNER/REPOSITORY")
                credentials {
                    username = project.findProperty("gpr.user") as String? ?:
System.getenv("USERNAME")
                    password = project.findProperty("gpr.key") as String? ?:
System.getenv("TOKEN")
                }
            }
        }
        publications {
            register<MavenPublication>("gpr") {
                from(components["java"])
            }
        }
    }
}
```

Publishing a package

By default, GitHub publishes the package to an existing repository with the same name as the package. For example, GitHub will publish a package named `com.example.test` in the `OWNER/test` GitHub Packages repository.

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

1. Authenticate to GitHub Packages. For more information, see "Authenticating to GitHub Packages."
2. After creating your package, you can publish the package.

```
gradle publish
```

Using a published package

To use a published package from GitHub Packages, add the package as a dependency and add the repository to your project. For more information, see "Declaring dependencies" in the Gradle documentation.

1. Authenticate to GitHub Packages. For more information, see "Authenticating to GitHub Packages."
2. Add the package dependencies to your *build.gradle* file (Gradle Groovy) or *build.gradle.kts* file (Kotlin DSL) file.

   Example using Gradle Groovy:

```
dependencies {
    implementation 'com.example:package'
}
```

   Example using Kotlin DSL:

```
dependencies {
    implementation("com.example:package")
}
```

3. Add the repository to your *build.gradle* file (Gradle Groovy) or *build.gradle.kts* file (Kotlin DSL) file.

   Example using Gradle Groovy:

```
repositories {
    maven {
        url = uri("https://REGISTRY_URL/OWNER/REPOSITORY")
        credentials {
            username = project.findProperty("gpr.user") ?:
System.getenv("USERNAME")
            password = project.findProperty("gpr.key") ?:
System.getenv("TOKEN")
        }
   }
}
```

Example using Kotlin DSL:

```
repositories {
    maven {
        url = uri("https://REGISTRY_URL/OWNER/REPOSITORY")
        credentials {
            username = project.findProperty("gpr.user") as String? ?:
System.getenv("USERNAME")
            password = project.findProperty("gpr.key") as String? ?:
System.getenv("TOKEN")
        }
    }
}
```

Further reading

- "Working with the Apache Maven registry"
- "Deleting and restoring a package"