Publishing Java packages with Gradle

In this article

You can use Gradle to publish Java packages to a registry as part of your continuous integration (CI) workflow.

Note: GitHub-hosted runners are not currently supported on GitHub Enterprise Server. You can see more information about planned future support on the GitHub public roadmap.

Introduction

This guide shows you how to create a workflow that publishes Java packages to GitHub Packages and the Maven Central Repository. With a single workflow, you can publish packages to a single repository or to multiple repositories.

Prerequisites

We recommend that you have a basic understanding of workflow files and configuration options. For more information, see "Learn GitHub Actions."

For more information about creating a CI workflow for your Java project with Gradle, see "Building and testing Java with Gradle."

You may also find it helpful to have a basic understanding of the following:

- "Working with the npm registry"
- "Variables"
- "Using secrets in GitHub Actions"
- "Automatic token authentication"

About package configuration

The `groupId` and `artifactId` fields in the `MavenPublication` section of the *build.gradle* file create a unique identifier for your package that registries use to link your package to a registry. This is similar to the `groupId` and `artifactId` fields of the Maven *pom.xml* file. For more information, see the "Maven Publish Plugin" in the Gradle documentation.

The *build.gradle* file also contains configuration for the distribution management repositories that Gradle will publish packages to. Each repository must have a name, a deployment URL, and credentials for authentication.

Publishing packages to the Maven Central Repository

Each time you create a new release, you can trigger a workflow to publish your package. The workflow in the example below runs when the `release` event triggers with type `created`. The workflow publishes the package to the Maven Central Repository if CI tests pass. For more information on the `release` event, see "Events that trigger workflows."

You can define a new Maven repository in the publishing block of your *build.gradle* file that points to your package repository. For example, if you were deploying to the Maven Central Repository through the OSSRH hosting project, your *build.gradle* could specify a repository with the name `"OSSRH"`.

Groovy

```
plugins {
  ...
  id 'maven-publish'
}

publishing {
  ...

  repositories {
    maven {
      name = "OSSRH"
      url = "https://oss.sonatype.org/service/local/staging/deploy/maven2/"
      credentials {
        username = System.getenv("MAVEN_USERNAME")
        password = System.getenv("MAVEN_PASSWORD")
      }
    }
  }
}
```

With this configuration, you can create a workflow that publishes your package to the Maven Central Repository by running the `gradle publish` command. In the deploy step, you'll need to set environment variables for the username and password or token that you use to authenticate to the Maven repository. For more information, see "Using secrets in GitHub Actions."

YAML

```
# This workflow uses actions that are not certified by GitHub.
# They are provided by a third-party and are governed by
# separate terms of service, privacy policy, and support
# documentation.

# GitHub recommends pinning actions to a commit SHA.
# To get a newer version, you will need to update the SHA.
# You can also reference a tag or branch, but the action may change without
warning.

name: Publish package to the Maven Central Repository
on:
  release:
    types: [created]
jobs:
  publish:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
```

```
      - name: Set up Java
        uses: actions/setup-java@v3
        with:
          java-version: '11'
          distribution: 'temurin'
      - name: Validate Gradle wrapper
        uses: gradle/wrapper-validation-
action@ccb4328a959376b642e027874838f60f8e596de3
      - name: Publish package
        uses: gradle/gradle-build-action@749f47bda3e44aa060e82d7b3ef7e40d953bd629
        with:
          arguments: publish
        env:
          MAVEN_USERNAME: ${{ secrets.OSSRH_USERNAME }}
          MAVEN_PASSWORD: ${{ secrets.OSSRH_TOKEN }}
```

This workflow performs the following steps:

1. Checks out a copy of project's repository.
2. Sets up the Java JDK.
3. Validates the checksums of any Gradle Wrapper JAR files present in the repository.
4. Runs the `gradle/gradle-build-action` action with the `publish` argument to publish to the `OSSRH` Maven repository. The `MAVEN_USERNAME` environment variable will be set with the contents of your `OSSRH_USERNAME` secret, and the `MAVEN_PASSWORD` environment variable will be set with the contents of your `OSSRH_TOKEN` secret.

   For more information about using secrets in your workflow, see "Using secrets in GitHub Actions."

Publishing packages to GitHub Packages

Each time you create a new release, you can trigger a workflow to publish your package. The workflow in the example below runs when the `release` event triggers with type `created`. The workflow publishes the package to GitHub Packages if CI tests pass. For more information on the `release` event, see "Events that trigger workflows."

You can define a new Maven repository in the publishing block of your *build.gradle* that points to GitHub Packages. In that repository configuration, you can also take advantage of environment variables set in your CI workflow run. You can use the `GITHUB_ACTOR` environment variable as a username, and you can set the `GITHUB_TOKEN` environment variable with your `GITHUB_TOKEN` secret.

The `GITHUB_TOKEN` secret is set to an access token for the repository each time a job in a workflow begins. You should set the permissions for this access token in the workflow file to grant read access for the `contents` scope and write access for the `packages` scope. For more information, see "Automatic token authentication."

For example, if your organization is named "octocat" and your repository is named "hello-world", then the GitHub Packages configuration in *build.gradle* would look similar to the below example.

Groovy

```
plugins {
  ...
  id 'maven-publish'
}

publishing {
```

```
  ...

  repositories {
    maven {
      name = "GitHubPackages"
      url = "https://maven.pkg.github.com/octocat/hello-world"
      credentials {
        username = System.getenv("GITHUB_ACTOR")
        password = System.getenv("GITHUB_TOKEN")
      }
    }
  }
}
```

With this configuration, you can create a workflow that publishes your package to GitHub Packages by running the `gradle publish` command.

YAML

```yaml

# This workflow uses actions that are not certified by GitHub.
# They are provided by a third-party and are governed by
# separate terms of service, privacy policy, and support
# documentation.

# GitHub recommends pinning actions to a commit SHA.
# To get a newer version, you will need to update the SHA.
# You can also reference a tag or branch, but the action may change without
warning.

name: Publish package to GitHub Packages
on:
  release:
    types: [created]
jobs:
  publish:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-java@v3
        with:
          java-version: '11'
          distribution: 'temurin'
      - name: Validate Gradle wrapper
        uses: gradle/wrapper-validation-
action@ccb4328a959376b642e027874838f60f8e596de3
      - name: Publish package
        uses: gradle/gradle-build-action@749f47bda3e44aa060e82d7b3ef7e40d953bd629
        with:
          arguments: publish
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

This workflow performs the following steps:

1. Checks out a copy of project's repository.
2. Sets up the Java JDK.
3. Validates the checksums of any Gradle Wrapper JAR files present in the repository.
4. Runs the `gradle/gradle-build-action` action with the `publish` argument to publish to GitHub Packages. The `GITHUB_TOKEN` environment variable will be set with the

content of the `GITHUB_TOKEN` secret. The `permissions` key specifies the access that the `GITHUB_TOKEN` secret will allow.

For more information about using secrets in your workflow, see "Using secrets in GitHub Actions."

Publishing packages to the Maven Central Repository and GitHub Packages

You can publish your packages to both the Maven Central Repository and GitHub Packages by configuring each in your *build.gradle* file.

Ensure your *build.gradle* file includes a repository for both your GitHub repository and your Maven Central Repository provider.

For example, if you deploy to the Central Repository through the OSSRH hosting project, you might want to specify it in a distribution management repository with the `name` set to `OSSRH`. If you deploy to GitHub Packages, you might want to specify it in a distribution management repository with the `name` set to `GitHubPackages`.

If your organization is named "octocat" and your repository is named "hello-world", then the configuration in *build.gradle* would look similar to the below example.

Groovy

```
plugins {
  ...
  id 'maven-publish'
}

publishing {
  ...

  repositories {
    maven {
      name = "OSSRH"
      url = "https://oss.sonatype.org/service/local/staging/deploy/maven2/"
      credentials {
        username = System.getenv("MAVEN_USERNAME")
        password = System.getenv("MAVEN_PASSWORD")
      }
    }
    maven {
      name = "GitHubPackages"
      url = "https://maven.pkg.github.com/octocat/hello-world"
      credentials {
        username = System.getenv("GITHUB_ACTOR")
        password = System.getenv("GITHUB_TOKEN")
      }
    }
  }
}
```

With this configuration, you can create a workflow that publishes your package to both the Maven Central Repository and GitHub Packages by running the `gradle publish` command.

YAML

```
# This workflow uses actions that are not certified by GitHub.
# They are provided by a third-party and are governed by
```

```
# separate terms of service, privacy policy, and support
# documentation.

# GitHub recommends pinning actions to a commit SHA.
# To get a newer version, you will need to update the SHA.
# You can also reference a tag or branch, but the action may change without
warning.

name: Publish package to the Maven Central Repository and GitHub Packages
on:
  release:
    types: [created]
jobs:
  publish:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write
    steps:
      - uses: actions/checkout@v4
      - name: Set up Java
        uses: actions/setup-java@v3
        with:
          java-version: '11'
          distribution: 'temurin'
      - name: Validate Gradle wrapper
        uses: gradle/wrapper-validation-
action@ccb4328a959376b642e027874838f60f8e596de3
      - name: Publish package
        uses: gradle/gradle-build-action@749f47bda3e44aa060e82d7b3ef7e40d953bd629
        with:
          arguments: publish
        env:
          MAVEN_USERNAME: ${{ secrets.OSSRH_USERNAME }}
          MAVEN_PASSWORD: ${{ secrets.OSSRH_TOKEN }}
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

This workflow performs the following steps:

1. Checks out a copy of project's repository.
2. Sets up the Java JDK.
3. Validates the checksums of any Gradle Wrapper JAR files present in the repository.
4. Runs the `gradle/gradle-build-action` action with the `publish` argument to publish to the `OSSRH` Maven repository and GitHub Packages. The `MAVEN_USERNAME` environment variable will be set with the contents of your `OSSRH_USERNAME` secret, and the `MAVEN_PASSWORD` environment variable will be set with the contents of your `OSSRH_TOKEN` secret. The `GITHUB_TOKEN` environment variable will be set with the content of the `GITHUB_TOKEN` secret. The `permissions` key specifies the access that the `GITHUB_TOKEN` secret will allow.

   For more information about using secrets in your workflow, see "Using secrets in GitHub Actions."