Publishing Java packages with Maven

In this article

You can use Maven 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 Maven, see "Building and testing Java with Maven."

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 *pom.xml* file create a unique identifier for your package that registries use to link your package to a registry. For more information see Guide to uploading artifacts to the Central Repository in the Apache Maven documentation.

The *pom.xml* file also contains configuration for the distribution management repositories that Maven will deploy packages to. Each repository must have a name and a deployment URL. Authentication for these repositories can be configured in the

.m2/settings.xml file in the home directory of the user running Maven.

You can use the `setup-java` action to configure the deployment repository as well as authentication for that repository. For more information, see [`setup-java`](setup-java).

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."

In this workflow, you can use the `setup-java` action. This action installs the given version of the JDK into the `PATH`, but it also configures a Maven *settings.xml* for publishing packages. By default, the settings file will be configured for GitHub Packages, but it can be configured to deploy to another package registry, such as the Maven Central Repository. If you already have a distribution management repository configured in *pom.xml*, then you can specify that `id` during the `setup-java` action invocation.

For example, if you were deploying to the Maven Central Repository through the OSSRH hosting project, your *pom.xml* could specify a distribution management repository with the `id` of `ossrh`.

XML

```xml
<project ...>
  ...
  <distributionManagement>
    <repository>
      <id>ossrh</id>
      <name>Central Repository OSSRH</name>
      <url>https://oss.sonatype.org/service/local/staging/deploy/maven2/</url>
    </repository>
  </distributionManagement>
</project>
```

With this configuration, you can create a workflow that publishes your package to the Maven Central Repository by specifying the repository management `id` to the `setup-java` action. You'll also need to provide environment variables that contain the username and password to authenticate to the repository.

In the deploy step, you'll need to set the environment variables to the username that you authenticate with to the repository, and to a secret that you've configured with the password or token to authenticate with. For more information, see "Using secrets in GitHub Actions."

YAML

```yaml
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 Maven Central Repository
        uses: actions/setup-java@v3
        with:
          java-version: '11'
          distribution: 'temurin'
          server-id: ossrh
          server-username: MAVEN_USERNAME
          server-password: MAVEN_PASSWORD
      - name: Publish package
        run: mvn --batch-mode deploy
        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, and also configures the Maven *settings.xml* file to add authentication for the `ossrh` repository using the `MAVEN_USERNAME` and `MAVEN_PASSWORD` environment variables.

3. Runs the `mvn --batch-mode deploy` command to publish to the `ossrh` 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."

In this workflow, you can use the `setup-java` action. This action installs the given version of the JDK into the `PATH`, and also sets up a Maven *settings.xml* for publishing the package to GitHub Packages. The generated *settings.xml* defines authentication for a server with an `id` of `github`, using the `GITHUB_ACTOR` environment variable as the username and the `GITHUB_TOKEN` environment variable as the password. The `GITHUB_TOKEN` environment variable is assigned the value of the special `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 a Maven-based project, you can make use of these settings by creating a distribution repository in your *pom.xml* file with an `id` of `github` that points to your GitHub Packages endpoint.

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

XML

```xml
<project ...>
  ...
  <distributionManagement>
    <repository>
```

```
      <id>github</id>
      <name>GitHub Packages</name>
      <url>https://maven.pkg.github.com/octocat/hello-world</url>
    </repository>
  </distributionManagement>
</project>
```

With this configuration, you can create a workflow that publishes your package to GitHub Packages by making use of the automatically generated *settings.xml*.

YAML

```yaml
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: Publish package
        run: mvn --batch-mode deploy
        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, and also automatically configures the Maven *settings.xml* file to add authentication for the `github` Maven repository to use the `GITHUB_TOKEN` environment variable.

3. Runs the `mvn --batch-mode deploy` command to publish to GitHub Packages. The `GITHUB_TOKEN` environment variable will be set with the contents of the `GITHUB_TOKEN` secret. The `permissions` key specifies the access granted to the `GITHUB_TOKEN`.

   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 using the `setup-java` action for each registry.

Ensure your *pom.xml* file includes a distribution management 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 `id` set to `ossrh`, and you might want to specify GitHub Packages in a distribution management repository with the `id` set to `github`.

YAML

```yaml
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 for publishing to Maven Central Repository
        uses: actions/setup-java@v3
        with:
          java-version: '11'
          distribution: 'temurin'
          server-id: ossrh
          server-username: MAVEN_USERNAME
          server-password: MAVEN_PASSWORD
      - name: Publish to the Maven Central Repository
        run: mvn --batch-mode deploy
        env:
          MAVEN_USERNAME: ${{ secrets.OSSRH_USERNAME }}
          MAVEN_PASSWORD: ${{ secrets.OSSRH_TOKEN }}
      - name: Set up Java for publishing to GitHub Packages
        uses: actions/setup-java@v3
        with:
          java-version: '11'
          distribution: 'temurin'
      - name: Publish to GitHub Packages
        run: mvn --batch-mode deploy
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

This workflow calls the `setup-java` action twice. Each time the `setup-java` action runs, it overwrites the Maven *settings.xml* file for publishing packages. For authentication to the repository, the *settings.xml* file references the distribution management repository `id`, and the username and password.

This workflow performs the following steps:

1. Checks out a copy of project's repository.

2. Calls `setup-java` the first time. This configures the Maven *settings.xml* file for the `ossrh` repository, and sets the authentication options to environment variables that are defined in the next step.

3. Runs the `mvn --batch-mode deploy` command to publish to the `ossrh` 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.

4. Calls `setup-java` the second time. This automatically configures the Maven *settings.xml* file for GitHub Packages.

5. Runs the `mvn --batch-mode deploy` command to publish to GitHub Packages. The `GITHUB_TOKEN` environment variable will be set with the contents of the `GITHUB_TOKEN` secret. The `permissions` key specifies the access granted to the `GITHUB_TOKEN`.

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