Publishing and installing a package with GitHub Actions

You can configure a workflow in GitHub Actions to automatically publish or install a package from GitHub Packages.

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."

In this article

About GitHub Packages with GitHub Actions

GitHub Actions help you automate your software development workflows in the same place you store code and collaborate on pull requests and issues. You can write individual tasks, called actions, and combine them to create a custom workflow. With GitHub Actions you can build end-to-end continuous integration (CI) and continuous deployment (CD) capabilities directly in your repository. For more information, see "Learn GitHub Actions."

You can extend the CI and CD capabilities of your repository by publishing or installing packages as part of your workflow.

Authenticating to package registries with granular permissions

Some GitHub Packages registries support granular permissions. This means you can choose to allow packages to be scoped to a user or an organization, or linked to a repository. For the list of registries that support granular permissions, see "About permissions for GitHub Packages."

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."

Authenticating to package registries with repository-scoped permissions

Some GitHub Packages registries only support repository-scoped permissions, and do not support granular permissions. For a list of these registries, see "About permissions for GitHub Packages."

If you want your workflow to access a GitHub Packages registry that does not support granular permissions, then we recommend using the `GITHUB_TOKEN` that GitHub Enterprise Cloud automatically creates for your repository when you enable GitHub Actions. 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 forks, the `GITHUB_TOKEN` is granted read access for the parent repository. For more information, see "Automatic token authentication."

You can reference the `GITHUB_TOKEN` in your workflow file using the `{{secrets.GITHUB_TOKEN}}` context. For more information, see "Automatic token authentication."

About permissions and package access

Packages scoped to users or organizations

Registries that support granular permissions allow users to create and administer packages as free-standing resources at the organization level. Packages can be scoped to an organization or personal account and you can customize access to each of your packages separately from repository permissions.

All workflows accessing registries that support granular permissions should use the `GITHUB_TOKEN` instead of a personal access token. For more information about security best practices, see "Security hardening for GitHub Actions."

Packages scoped to repositories

When you enable GitHub Actions, GitHub installs a GitHub App on your repository. The `GITHUB_TOKEN` secret is a GitHub App installation access token. You can use the installation access token to authenticate on behalf of the GitHub App installed on your repository. The token's permissions are limited to the repository that contains your workflow. For more information, see "Automatic token authentication."

GitHub Packages allows you to push and pull packages through the `GITHUB_TOKEN` available to a GitHub Actions workflow.

Default permissions and access settings for packages modified through workflows

For packages in registries that support granular permissions, when you create, install, modify, or delete a package through a workflow, there are some default permission and access settings used to ensure admins have access to the workflow. You can adjust these access settings as well. For the list of registries that support granular permissions, see "About permissions for GitHub Packages."

For example, by default if a workflow creates a package using the `GITHUB_TOKEN`, then:

- The package inherits the visibility and permissions model of the repository where the workflow is run.
- Repository admins where the workflow is run become the admins of the package once the package is created.

These are more examples of how default permissions work for workflows that manage packages.

GitHub Actions workflow task	Default permissions and access
Download an existing	- If the package is public, any workflow running in any repository can download the package. - If the package is internal, then all workflows running in any repository owned by the Enterprise account can download the package. For enterprise-owned organizations, you can read any repository in the enterprise - If the package is private, only workflows running in repositories that are given read permission on that package can download the package.
Upload a new version to an existing package	- If the package is private, internal, or public, only workflows running in repositories that are given write permission on that package can upload new versions to the package.
Delete a package or versions of a package	- If the package is private, internal, or public, only workflows running in repositories that are given admin permission can delete existing versions of the package.

You can also adjust access to packages in a more granular way or adjust some of the default permissions behavior. For more information, see "Configuring a package's access control and visibility."

Publishing a package using an action

You can use GitHub Actions to automatically publish packages as part of your continuous integration (CI) flow. This approach to continuous deployment (CD) allows you to automate the creation of new package versions, if the code meets your quality standards. For example, you could create a workflow that runs CI tests every time a developer pushes code to a particular branch. If the tests pass, the workflow can publish a new package version to GitHub Packages.

Configuration steps vary by package client. For general information about configuring a workflow for GitHub Actions, see "Using workflows."

The following example demonstrates how you can use GitHub Actions to build and test your app, and then automatically create a Docker image and publish it to GitHub Packages. The relevant settings are explained in the code. For full details about each element in a workflow, see "Workflow syntax for GitHub Actions."

Create a new workflow file in your repository (such as `.github/workflows/deploy-image.yml`), and add the following YAML.

Notes:

- 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.

YAML Beside Inline

```yaml
name: Create and publish a Docker image
```

```yaml
on:
  push:
    branches: ['release']
```

Configures this workflow to run every time a change is pushed to the branch called `release`.

```yaml
env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}
```

Defines two custom environment variables for the workflow. These are used for the Container registry domain, and a name for the Docker image that this workflow builds.

```yaml
jobs:
  build-and-push-image:
    runs-on: ubuntu-latest
```

There is a single job in this workflow. It's configured to run on the latest available version of Ubuntu.

```yaml
    permissions:
      contents: read
      packages: write
```

Sets the permissions granted to the `GITHUB_TOKEN` for the actions in this job.

```yaml
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4
      - name: Log in to the Container registry
        uses: docker/login-action@65b78e6e13532edd9afa3aa52ac7964289d1a9c1
        with:
          registry: ${{ env.REGISTRY }}
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}
```

Uses the `docker/login-action` action to log in to the Container registry registry using the account and password that will publish the packages. Once published, the packages are scoped to the account defined here.

```
      - name: Extract metadata (tags, labels) for Docker
        id: meta
        uses: docker/metadata-action@9ec57ed1fcdbf14dcef7dfbe97b2010124a938b7
        with:
          images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
```

This step uses docker/metadata-action to extract tags and labels that will be applied to the specified image. The `id` "meta" allows the output of this step to be referenced in a subsequent step. The `images` value provides the base name for the tags and labels.

```
      - name: Build and push Docker image
        uses: docker/build-push-
action@f2a1d5e99d037542a71f64918e516c093c6f3fc4
        with:
          context: .
          push: true
          tags: ${{ steps.meta.outputs.tags }}
          labels: ${{ steps.meta.outputs.labels }}
```

This step uses the `docker/build-push-action` action to build the image, based on your repository's `Dockerfile`. If the build succeeds, it pushes the image to GitHub Packages. It uses the `context` parameter to define the build's context as the set of files located in the specified path. For more information, see "Usage" in the README

of the `docker/build-push-action` repository. It uses the `tags` and `labels` parameters to tag and label the image with the output from the "meta" step.

This new workflow will run automatically every time you push a change to a branch named `release` in the repository. You can view the progress in the **Actions** tab.

A few minutes after the workflow has completed, the new package will visible in your repository. To find your available packages, see "Viewing packages."

Installing a package using an action

You can install packages as part of your CI flow using GitHub Actions. For example, you could configure a workflow so that anytime a developer pushes code to a pull request, the workflow resolves dependencies by downloading and installing packages hosted by GitHub Packages. Then, the workflow can run CI tests that require the dependencies.

Installing packages hosted by GitHub Packages through GitHub Actions requires minimal configuration or additional authentication when you use the `GITHUB_TOKEN`. Data transfer is also free when an action installs a package. For more information, see "About billing for GitHub Packages."

Configuration steps vary by package client. For general information about configuring a workflow for GitHub Actions, see "Using workflows."

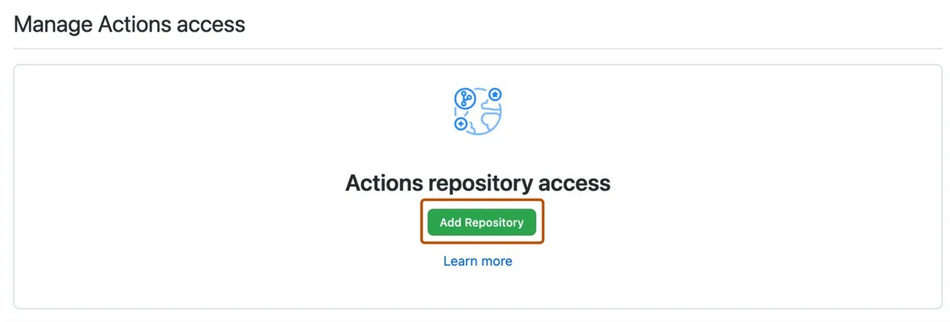
Upgrading a workflow that accesses a registry using a personal access token

GitHub Packages supports the `GITHUB_TOKEN` for easy and secure authentication in your workflows. If you're using a registry that supports granular permissions, and your workflow is using a personal access token to authenticate to the registry, then we highly recommend you update your workflow to use the `GITHUB_TOKEN`.

For more information about the `GITHUB_TOKEN`, see "Automatic token authentication."

Using the `GITHUB_TOKEN`, instead of a personal access token (classic) with the `repo` scope, increases the security of your repository as you don't need to use a long-lived personal access token that offers unnecessary access to the repository where your workflow is run. For more information about security best practices, see "Security hardening for GitHub Actions."

1. Navigate to your package landing page.
2. To ensure your package has access to your workflow, you must add the repository where the workflow is stored to your package. Under "Manage Actions access", click **Add repository** and search for the repository you want to add.



Note: Adding a repository to your package by using the **Add Repository** button under "Manage Actions access" in the package's settings is different than connecting your package to a repository. For more information, see "Configuring a package's access control and visibility" and "Connecting a repository to a package."

3. Optionally, use the **Role** drop-down menu to select the default access level that you'd like the repository to have to your package.
4. Open your workflow file. On the line where you log in to the registry, replace your personal access token with `${{ secrets.GITHUB_TOKEN }}`.

For example, this workflow publishes a Docker image to the Container registry and uses `${{ secrets.GITHUB_TOKEN }}` to authenticate. For more information, see "Set up Automated Builds" in the Docker documentation.

YAML   Beside   Inline

```yaml
name: Demo Push
```

```
on:
  push:
    branches:
      - main
      - seed
    tags:
      - v*
  pull_request:
```

This workflow runs when any of the following occur:

- A push is made to a branch called `main` or `seed`
- A tag starting with "v" is created
- A pull request is created or updated

```
env:
  IMAGE_NAME: ghtoken_product_demo
```

This creates an environment variable called `IMAGE_NAME` with the value `ghtoken_product_demo`.

```
jobs:
```

```
  push:
    runs-on: ubuntu-latest
    permissions:
      packages: write
      contents: read
```

This pushes the image to GitHub Packages.

```
    steps:
      - uses: actions/checkout@v4
      - name: Build image
        run: docker build . --file Dockerfile --tag $IMAGE_NAME --label
"runnumber=${GITHUB_RUN_ID}"
      - name: Log in to registry
        run: echo "${{ secrets.GITHUB_TOKEN }}" | docker login ghcr.io -u $ -
-password-stdin
```

```
      - name: Push image
        run: |
          IMAGE_ID=ghcr.io/${{ github.repository_owner }}/$IMAGE_NAME
```

```
          IMAGE_ID=$(echo $IMAGE_ID | tr '[A-Z]' '[a-z]')
```

This changes all uppercase characters to lowercase.

```
          VERSION=$(echo "${{ github.ref }}" | sed -e 's,.*/\(.*\),\1,')
```

This strips the git ref prefix from the version.

This strips the git ref prefix from the version.

```
          [[ "${{ github.ref }}" == "refs/tags/"* ]] && VERSION=$(echo
$VERSION | sed -e 's/^v//')
```

This strips the "v" prefix from the tag name.

```
          [ "$VERSION" == "main" ] && VERSION=latest
          echo IMAGE_ID=$IMAGE_ID
          echo VERSION=$VERSION
          docker tag $IMAGE_NAME $IMAGE_ID:$VERSION
          docker push $IMAGE_ID:$VERSION
```

This uses the Docker `latest` tag convention.