Working with the Docker registry

In this article

You can push and pull your Docker images using the GitHub Packages Docker registry.

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."

About Docker support

When installing or publishing a Docker image, the Docker registry does not currently support foreign layers, such as Windows images.

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."

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 Docker using the `docker` login command.

To keep your credentials secure, we recommend you save your personal access token in a local file on your computer and use Docker's `--password-stdin` flag, which reads your token from a local file.

If your instance has subdomain isolation enabled:

```
cat ~/TOKEN.txt | docker login docker.HOSTNAME -u USERNAME --password-stdin
```

If your instance has subdomain isolation disabled:

```
cat ~/TOKEN.txt | docker login HOSTNAME -u USERNAME --password-stdin
```

To use this example login command, replace `USERNAME` with your GitHub Enterprise Server username, `HOSTNAME` with the URL for your GitHub Enterprise Server instance, and `~/TOKEN.txt` with the file path to your personal access token for GitHub Enterprise Server.

For more information, see "Docker login."

Publishing an image

Note: The GitHub Packages Docker registry will be superseded in a future GitHub Enterprise Server release with the Container registry, which offers improved container support.

Note: Image names must only use lowercase letters.

GitHub Packages supports multiple top-level Docker images per repository. A repository can have any number of image tags. You may experience degraded service publishing or installing Docker images larger than 10GB, layers are capped at 5GB each. For more information, see "Docker tag" in the Docker documentation.

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

1. Determine the image name and ID for your docker image using `docker images`.

   ```
   $ docker images
   > < >
   > REPOSITORY        TAG         IMAGE ID       CREATED      SIZE
   > IMAGE_NAME        VERSION     IMAGE_ID       4 weeks ago  1.11MB
   ```

2. Using the Docker image ID, tag the docker image, replacing OWNER with the name of the personal account or organization that owns the repository, REPOSITORY with the name of the repository containing your project, IMAGE_NAME with name of the package or image, HOSTNAME with the hostname of your GitHub Enterprise Server instance, and VERSION with package version at build time.

   If your instance has subdomain isolation enabled:

   ```
   docker tag IMAGE_ID docker.HOSTNAME/OWNER/REPOSITORY/IMAGE_NAME:VERSION
   ```

If your instance has subdomain isolation disabled:

```
docker tag IMAGE_ID HOSTNAME/OWNER/REPOSITORY/IMAGE_NAME:VERSION
```

3. If you haven't already built a docker image for the package, build the image, replacing OWNER with the name of the personal account or organization that owns the repository, REPOSITORY with the name of the repository containing your project, IMAGE_NAME with name of the package or image, VERSION with package version at build time, HOSTNAME with the hostname of your GitHub Enterprise Server instance, and PATH to the image if it isn't in the current working directory.

   If your instance has subdomain isolation enabled:

   ```
   docker build -t docker.HOSTNAME/OWNER/REPOSITORY/IMAGE_NAME:VERSION PATH
   ```

   If your instance has subdomain isolation disabled:

   ```
   docker build -t HOSTNAME/OWNER/REPOSITORY/IMAGE_NAME:VERSION PATH
   ```

4. Publish the image to GitHub Packages.

   If your instance has subdomain isolation enabled:

   ```
   docker push docker.HOSTNAME/OWNER/REPOSITORY/IMAGE_NAME:VERSION
   ```

   If your instance has subdomain isolation disabled:

   ```
   docker push HOSTNAME/OWNER/REPOSITORY/IMAGE_NAME:VERSION
   ```

   Note: You must push your image using `IMAGE_NAME:VERSION` and not using `IMAGE_NAME:SHA`.

Example publishing a Docker image

These examples assume your instance has subdomain isolation enabled.

You can publish version 1.0 of the `monalisa` image to the `octocat/octo-app` repository using an image ID.

```
$ docker images

> REPOSITORY           TAG      IMAGE ID      CREATED      SIZE
> monalisa             1.0      c75bebcdd211  4 weeks ago  1.11MB

# Tag the image with OWNER/REPO/IMAGE_NAME
$ docker tag c75bebcdd211 docker.HOSTNAME/octocat/octo-app/monalisa:1.0

# Push the image to GitHub Packages
$ docker push docker.HOSTNAME/octocat/octo-app/monalisa:1.0
```

You can publish a new Docker image for the first time and name it `monalisa`.

```
# Build the image with docker.HOSTNAME/OWNER/REPOSITORY/IMAGE_NAME:VERSION
# Assumes Dockerfile resides in the current working directory (.)
$ docker build -t docker.HOSTNAME/octocat/octo-app/monalisa:1.0 .
```

```
# Push the image to GitHub Packages
$ docker push docker.HOSTNAME/octocat/octo-app/monalisa:1.0
```

Downloading an image

Note: The GitHub Packages Docker registry will be superseded in a future GitHub Enterprise Server release with the Container registry, which offers improved container support.

You can use the `docker pull` command to install a docker image from GitHub Packages, replacing OWNER with the name of the personal account or organization that owns the repository, REPOSITORY with the name of the repository containing your project, IMAGE_NAME with name of the package or image, HOSTNAME with the host name of your GitHub Enterprise Server instance, and TAG_NAME with tag for the image you want to install.

If your instance has subdomain isolation enabled:

```
docker pull docker.HOSTNAME/OWNER/REPOSITORY/IMAGE_NAME:TAG_NAME
```

If your instance has subdomain isolation disabled:

```
docker pull HOSTNAME/OWNER/REPOSITORY/IMAGE_NAME:TAG_NAME
```

Note: You must pull the image using `IMAGE_NAME:VERSION` and not using `IMAGE_NAME:SHA`.

Further reading

- "Deleting and restoring a package"