Red Hat Developer Hub 1.8

Integrating Red Hat Developer Hub with GitHub

Configuring integration to the GitHub Git provider in Red Hat Developer Hub

Last Updated: 2026-01-22

Red Hat Developer Hub 1.8 Integrating Red Hat Developer Hub with GitHub

Configuring integration to the GitHub Git provider in Red Hat Developer Hub

Legal Notice

Abstract

As a Red Hat Developer Hub (RHDH) platform engineer, you can integrate RHDH with the GitHub Git provider.

Table of Contents

CHAPTER 1. ENABLING GITHUB REPOSITORY DISCOVERY

Consider configuring Developer Hub to discover and ingest your GitHub repositories automatically. If a repository contains a **catalog-info.yaml** file, Developer Hub ingests the repository into the catalog as a component.

Prerequisites

- You added a custom Developer Hub application configuration , and have sufficient permissions to modify it.
- You have sufficient permissions in GitHub to create and manage a GitHub App.
- To allow users to access GitHub templates or plugins that require GitHub authentication, you have configured GitHub either as an auxiliary authentication provider or as your main authentication provider.

Procedure

1. To allow Developer Hub to access the GitHub API, create a GitHub App. Opt for a GitHub App instead of an OAuth app to use fine-grained permissions, gain more control over which repositories the application can access, and use short-lived tokens.
   a. Register a GitHub App with the following configuration:

      GitHub App name

      Enter a unique name identifying your GitHub App, such as **integrating-with-rhdh-*<GUID>***.

      Homepage URL

      Enter your Developer Hub URL: **https://*<my_developer_hub_domain>***.

      Authorization callback URL

      Enter your Developer Hub authentication backend URL: **https://*<my_developer_hub_domain>*/api/auth/github/handler/frame**.

      Webhook

      Clear "Active", as this is not needed for authentication and catalog providers.

      App permissions

      Select permissions to define the level of access for the app. Adapt permissions to your needs:

      Reading software components

      Contents

      Read-only

      Commit statuses

      Read-only

      Reading organization data

      Members

      Read-only

      Publishing software templates

Set permissions if you intend to use the same GitHub App for software templates.

Administration

Read & write (for creating repositories)

Contents

Read & write

Metadata

Read-only

Pull requests

Read & write

Issues

Read & write

Workflows

Read & write (if templates include GitHub workflows)

Variables

Read & write (if templates include GitHub Action Repository Variables)

Secrets

Read & write (if templates include GitHub Action Repository Secrets)

Environments

Read & write (if templates include GitHub Environments)

Organization permissions

Members

Read-only

Where can this GitHub App be installed?

Select **Only on this account**.

b. In the **General → Clients secrets** section, click **Generate a new client secret**

c. In the **General → Private keys** section, click **Generate a private key**.

d. In the **Install App** tab, choose an account to install your GitHub App on.

e. Save the following values for the next step:

- **App ID**
- **Client ID**
- **Client secret**
- **Private key**

2. To add your GitHub credentials to Developer Hub, add the following key/value pairs to your Developer Hub secrets. You can use these secrets in the Developer Hub configuration files by using their respective environment variable name.

GITHUB_INTEGRATION_APP_ID

Enter the saved **App ID**.

GITHUB_INTEGRATION_CLIENT_ID

Enter the saved **Client ID**.

GITHUB_INTEGRATION_CLIENT_SECRET

Enter the saved **Client Secret**.

GITHUB_INTEGRATION_HOST_DOMAIN

Enter the GitHub host domain: **github.com**.

GITHUB_INTEGRATION_ORGANIZATION

Enter your GitHub organization name, such as `*<your_github_organization_name>*'.

GITHUB_INTEGRATION_PRIVATE_KEY_FILE

Enter the saved **Private key**.

3. Enable the **plugin-catalog-backend-module-github** plugin in your **dynamic-plugins.yaml** file. This plugin discovers catalog entities by scanning repositories within a GitHub organization for **catalog-info.yaml** files. It provides an automated alternative to manually registering components via **catalog.locations**. When a repository contains a **catalog-info.yaml** file, the entity is ingested into the catalog as a component.

dynamic-plugins.yaml file fragment

```
plugins:
  - package: './dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github'
    disabled: false
```

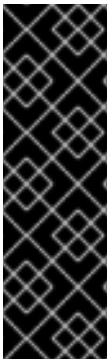
4. Configure the GitHub integration, by adding the **catalog.providers.github** and the **integrations.github** sections to your custom Developer Hub **app-config.yaml** configuration file:

app-config.yaml file fragment with mandatory fields to enable GitHub integration

```
catalog:
  providers:
    github:
      providerId:
        organization: "${GITHUB_INTEGRATION_ORGANIZATION}"
        schedule:
          frequency:
            minutes: 30
          initialDelay:
            seconds: 15
          timeout:
            minutes: 15
integrations:
  github:
    - host: ${GITHUB_INTEGRATION_HOST_DOMAIN}
      apps:
        - appId: ${GITHUB_INTEGRATION_APP_ID}
          clientId: ${GITHUB_INTEGRATION_CLIENT_ID}
```

```
    clientSecret: ${GITHUB_INTEGRATION_CLIENT_SECRET}
    privateKey: |
      ${GITHUB_INTEGRATION_PRIVATE_KEY_FILE}
```

CHAPTER 2. BULK IMPORTING IN RED HAT DEVELOPER HUB



IMPORTANT

These features are for Technology Preview only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend using them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information on Red Hat Technology Preview features, see Technology Preview Features Scope.

Red Hat Developer Hub can automate the onboarding of GitHub repositories and GitLab projects, and track their import status.

2.1. ENABLING AND AUTHORIZING BULK IMPORT CAPABILITIES IN RED HAT DEVELOPER HUB

You can enable the Bulk Import feature for users and give them the necessary permissions to access it. This feature is available for GitHub repositories and GitLab projects.

Prerequisites

- For GitHub only: You have enabled GitHub repository discovery.

Procedure

1. The Bulk Import plugins are installed but disabled by default. To enable the **./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import-backend-dynamic** and **./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import** plugins, edit your **dynamic-plugins.yaml** with the following content:

```
plugins:
  - package: ./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import-backend-dynamic
    disabled: false
  - package: ./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import
    disabled: false
```

   See Installing and viewing plugins in Red Hat Developer Hub .

2. Configure the required **bulk.import** RBAC permission for the users who are not administrators as shown in the following code:

   rbac-policy.csv fragment

```
p, role:default/bulk-import, bulk.import, use, allow
g, user:default/<your_user>, role:default/bulk-import
```

   Note that only Developer Hub administrators or users with the **bulk.import** permission can use the Bulk Import feature. See Permission policies in Red Hat Developer Hub .

Verification

- The sidebar displays a **Bulk Import** option.
- The **Bulk Import** page shows a list of added GitHub repositories and GitLab projects.

2.2. IMPORTING MULTIPLE GITHUB REPOSITORIES

In Red Hat Developer Hub, you can select your GitHub repositories and automate their onboarding to the Developer Hub catalog.

Prerequisites

- You have enabled the Bulk Import feature and gave access to it .

Procedure

1. Click **Bulk Import** in the left sidebar.
2. Click the **Add** button in the top-right corner to see the list of all repositories accessible from the configured GitHub integrations.
   a. From the **Repositories** view, you can select any repository, or search for any accessible repositories. For each repository selected, a **catalog-info.yaml** is generated.
   b. From the **Organizations** view, you can select any organization by clicking **Select** in the third column. This option allows you to select one or more repositories from the selected organization.
3. Click **Preview file** to view or edit the details of the pull request for each repository.
   a. Review the pull request description and the **catalog-info.yaml** file content.
   b. Optional: when the repository has a **.github/CODEOWNERS** file, you can select the **Use CODEOWNERS file as Entity Owner** checkbox to use it, rather than having the **content-info.yaml** contain a specific entity owner.
   c. Click **Save**.
4. Click **Create pull requests**. At this point, a set of dry-run checks runs against the selected repositories to ensure they meet the requirements for import, such as:
   a. Verifying that there is no entity in the Developer Hub catalog with the name specified in the repository **catalog-info.yaml**
   b. Verifying that the repository is not empty
   c. Verifying that the repository contains a **.github/CODEOWNERS** file if the **Use CODEOWNERS file as Entity Owner** checkbox is selected for that repository
      - If any errors occur, the pull requests are not created, and you see a *Failed to create PR* error message detailing the issues. To view more details about the reasons, click **Edit**.
      - If there are no errors, the pull requests are created, and you are redirected to the list of added repositories.

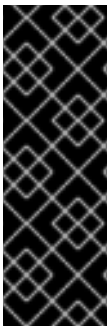
5. Review and merge each pull request that creates a **catalog-info.yml** file.

Verification

- The **Added entities** list displays the repositories you imported, each with an appropriate status: either *Waiting for approval* or *Added*.
- For each *Waiting for approval* import job listed, there is a corresponding pull request adding the **catalog-info.yaml** file in the corresponding repository.

2.3. IMPORTING MULTIPLE GITLAB REPOSITORIES

In Red Hat Developer Hub, you can select your GitLab projects and automate their onboarding to the Developer Hub catalog. This feature is a Technology preview.



IMPORTANT

Technology Preview features provide early access to upcoming product innovations, enabling you to test functionality and provide feedback during the development process. However, these features are not fully supported under Red Hat Subscription Level Agreements, may not be functionally complete, and are not intended for production use. As Red Hat considers making future iterations of Technology Preview features generally available, we will attempt to resolve any issues that customers experience when using these features. See: Technology Preview support scope.

Prerequisites

- You have enabled the Bulk Import feature and given access to it .
- You have set up a GitLab personal access token (PAT).

Procedure

1. In RHDH, click **Bulk Import**
2. Click **Import**.
3. Select GitLab as your **Approval tool** option.
4. Use the **Project** and **Group** views to see the list of all available GitLab projects and groups:
   - Use the **Project** view to select GitLab projects for importing.
   - Use the **Group** view to select GitLab groups and their associated projects for importing.
5. In GitLab, review the automatically created "Ad **catalog-info.yaml** file" merge request for each project you selected for Bulk Import.
6. Merge the merge request.

Verification

1. In RHDH, click **Bulk Import**

2. In the **Imported entities** list, each imported GitLab project has the appropriate status: either *Waiting for approval* or *Added*.
    - For each *Waiting for approval* import job listed, there is a corresponding merge request adding the **catalog-info.yaml** file in the corresponding project.

2.4. MONITORING BULK IMPORT ACTIONS USING AUDIT LOGS

The Bulk Import backend plugin adds the following events to the Developer Hub audit logs. See Audit logs in Red Hat Developer Hub for more information on how to configure and view audit logs.

Bulk Import Events:

BulkImportUnknownEndpoint

Tracks requests to unknown endpoints.

BulkImportPing

Tracks **GET** requests to the **/ping** endpoint, which allows us to make sure the bulk import backend is up and running.

BulkImportFindAllOrganizations

Tracks **GET** requests to the **/organizations** endpoint, which returns the list of organizations accessible from all configured GitHub Integrations.

BulkImportFindRepositoriesByOrganization

Tracks **GET** requests to the **/organizations/:orgName/repositories** endpoint, which returns the list of repositories for the specified organization (accessible from any of the configured GitHub Integrations).

BulkImportFindAllRepositories

Tracks GET requests to the **/repositories** endpoint, which returns the list of repositories accessible from all configured GitHub Integrations.

BulkImportFindAllImports

Tracks **GET** requests to the **/imports** endpoint, which returns the list of existing import jobs along with their statuses.

BulkImportCreateImportJobs

Tracks **POST** requests to the **/imports** endpoint, which allows to submit requests to bulk-import one or many repositories into the Developer Hub catalog, by eventually creating import pull requests in the target repositories.

BulkImportFindImportStatusByRepo

Tracks **GET** requests to the **/import/by-repo** endpoint, which fetches details about the import job for the specified repository.

BulkImportDeleteImportByRepo

Tracks **DELETE** requests to the **/import/by-repo** endpoint, which deletes any existing import job for the specified repository, by closing any open import pull request that could have been created.

Example bulk import audit logs


```
{
  "actor": {
    "actorId": "user:default/myuser",
    "hostname": "localhost",
    "ip": "::1",
```

```
    "userAgent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/128.0.0.0 Safari/537.36"
  },
  "eventName": "BulkImportFindAllOrganizations",
  "isAuditLog": true,
  "level": "info",
  "message": "'get /organizations' endpoint hit by user:default/myuser",
  "meta": {},
  "plugin": "bulk-import",
  "request": {
    "body": {},
    "method": "GET",
    "params": {},
    "query": {
      "pagePerIntegration": "1",
      "sizePerIntegration": "5"
    },
    "url": "/api/bulk-import/organizations?pagePerIntegration=1&sizePerIntegration=5"
  },
  "response": {
    "status": 200
  },
  "service": "backstage",
  "stage": "completion",
  "status": "succeeded",
  "timestamp": "2024-08-26 16:41:02"
}
```


2.5. INPUT PARAMETERS FOR BULK IMPORT SCAFFOLDER TEMPLATE

As an administrator, you can use the Bulk Import plugin to run a Scaffolder template task with specified parameters, which you must define within the template.

The Bulk Import plugin analyzes Git repository information and provides the following parameters for the Scaffolder template task:

repoUrl

Normalized repository URL in the following format:

```
${gitProviderHost}?owner=${owner}&repo=${repository-name}
```

name

The repository name.

organization

The repository owner, which can be a user nickname or organization name.

branchName

The proposed repository branch. By default, the proposed repository branch is **bulk-import-catalog-entity**.

targetBranchName

The default branch of the Git repository.

gitProviderHost

The Git provider host parsed from the repository URL. You can use this parameter to write **Git-provider-agnostic** templates.

Example of a Scaffolder template:

```
parameters:
  - title: Repository details
    required:
      - repoUrl
      - branchName
      - targetBranchName
      - name
      - organization
    properties:
      repoUrl:
        type: string
        title: Repository URL (Backstage format)
        description: github.com?owner=Org&repo=repoName
      organization:
        type: string
        title: Owner of the repository
      name:
        type: string
        title: Name of the repository
      branchName:
        type: string
        title: Branch to add the catalog entity to
      targetBranchName:
        type: string
        title: Branch to target the PR/MR to
      gitProviderHost:
        type: string
        title: Git provider host
```

2.6. SETTING UP A CUSTOM SCAFFOLDER WORKFLOW FOR BULK IMPORT

As an administrator, you can create a custom Scaffolder template in line with the repository conventions of your organization and add the template into the Red Hat Developer Hub catalog for use by the Bulk Import plugin on multiple selected repositories.

You can define various custom tasks, including, but not limited to the following:

- Importing existing catalog entities from a repository
- Creating pull requests for cleanup
- Calling webhooks for external system integration

Prerequisites

- You created a custom Scaffolder template for the Bulk Import plugin.

- You have run your RHDH instance with the following environment variable enabled to allow the use of the Scaffolder functionality:

```
export NODE_OPTIONS=--no-node-snapshot
```

Procedure

- Configure your app-config.yaml configuration to instruct the Bulk Import plugin to use your custom template as shown in the following example:

```
bulkImport:
  importTemplate: <your_template_entity_reference_or_template_name>
  importAPI: `open-pull-requests` | `scaffolder`;
```

where:

importTemplate:

Enter your Scaffolder template entity reference.

importAPI

Set the API to 'scaffolder' to trigger the defined workflow for high-fidelity automation. This field defines the import workflow and currently supports two following options:

open-pull-requests

This is the default import workflow, which includes the logic for creating pull requests for every selected repository.

scaffolder

This workflow uses an import scenario defined in the Scaffolder template to create import jobs. Select this option to use the custom import scenario defined in your Scaffolder template.
Optional: You can direct the Bulk Import plugin to hand off the entire list of selected repositories to a custom Orchestrator workflow.

IMPORTANT

The Scaffolder template must be generic and not specific to a single repository if you want your custom Scaffolder template to run successfully for every repository in the bulk list.

Verification

- The Bulk Import plugin runs the custom Scaffolder template for the list of repositories using the **/task-imports** API endpoint.

2.7. DATA HANDOFF AND CUSTOM WORKFLOW DESIGN

When you configure the Bulk Import plugin by setting the importAPI field to **scaffolder**, the Bulk Import Backend passes all necessary context directly to the Scaffolder API.

As an administrator, you can define the Scaffolder template workflow and structure the workflow to do the following:

Define template parameters to consume input

Structure the Scaffolder template to receive the repository data as template parameters for the current workflow run. The template must be generic, and not specific to a single repository, so that it can successfully run for every repository in the bulk list.

Automate processing for each repository

Implement the custom logic needed for a single repository within the template. The Orchestrator iterates through the repository list, launching the template once for each repository and passes only the data for that single repository to the template run. This allows you to automate tasks such as creating the **catalog-info.yaml**, running compliance checks, or registering the entity with the catalog.