Red Hat Developer Hub 1.8

Setting up and configuring your first Red Hat Developer Hub instance

Prepare your IT infrastructure including Red Hat OpenShift Container Platform and required external components, and run your first Red Hat Developer Hub (RHDH) instance in production.

Last Updated: 2026-01-22

Red Hat Developer Hub 1.8 Setting up and configuring your first Red Hat Developer Hub instance

Prepare your IT infrastructure including Red Hat OpenShift Container Platform and required external components, and run your first Red Hat Developer Hub (RHDH) instance in production.

Legal Notice

Abstract

Prepare your IT infrastructure including Red Hat OpenShift Container Platform and required external components, and run your first Red Hat Developer Hub (RHDH) instance in production.

Table of Contents

PREFACE

Prepare your IT infrastructure including Red Hat OpenShift Container Platform and required external components, and run your first Red Hat Developer Hub (RHDH) instance in production.

CHAPTER 1. CHECKLIST TO RUN YOUR FIRST RED HAT DEVELOPER HUB (RHDH) INSTANCE IN PRODUCTION.

With the default configuration, Developer Hub runs with a minimal feature set that does not require secure connection to external services such as an identity provider, a Git provider, and external PostgreSQL and Redis databases.

Using critical features therefore requires following additional configuration:

For resiliency

- Use an external PostgreSQL database.
- Enable high-availability.

For performance

- Enable assets caching to an external Redis database.

For security

- Use secure connections to your external services.
- Provision users and enable authentication.
- Enable role-based access control, and configure the permission policy by using the Web UI.

For adapting to your environment

- Enable GitHub repository discovery.
- Customize Developer Hub appearance with your logo.

CHAPTER 2. INSTALLING THE RED HAT DEVELOPER HUB OPERATOR

As an administrator, you can install the Red Hat Developer Hub Operator. Authorized users can use the Operator to install Red Hat Developer Hub on Red Hat OpenShift Container Platform (OpenShift Container Platform) and supported Kubernetes platforms. For more information on supported platforms and versions, see the Red Hat Developer Hub Life Cycle page.

Containers are available for the following CPU architectures:

- AMD64 and Intel 64 (**x86_64**)

Prerequisites

- You are logged in as an administrator on the OpenShift Container Platform web console.
- You have configured the appropriate roles and permissions within your project to create or access an application. For more information, see the Red Hat OpenShift Container Platform documentation on Building applications.
- You have installed Red Hat OpenShift Container Platform 4.16 to 4.19.
- Make sure that your system meets the minimum sizing requirements. See Sizing requirements for Red Hat Developer Hub.

Procedure

1. In the navigation menu of the OpenShift Container Platform console, click **Operators > OperatorHub**.
2. In the **Filter by keyword** box, enter Developer Hub and click the **Red Hat Developer Hub Operator** card.
3. On the **Red Hat Developer Hub Operator**page, read the information about the Operator and click **Install** to open the **Install Operator** page.
4. After the Operator is successfully installed, provision your custom configuration:
   Before you create a Developer Hub instance, you must create the required config map and Secret resources in your project. These include the **baseUrl** and service-to-service authentication secrets.

   For detailed steps, see Provisioning your custom Red Hat Developer Hub configuration .
5. From the **Update channel** drop-down menu, select the update channel that you want to use, for example, **fast** or **fast-1.8**.

IMPORTANT

The `**fast** channel includes all of the updates available for a particular version. Any update might introduce unexpected changes in your Red Hat Developer Hub deployment. Check the release notes for details about any potentially breaking changes.

The **fast-1.8** channel only provides z-stream updates, for example, updating from version 1.8.1 to 1.8.2. If you want to update the Red Hat Developer Hub y-version in the future, for example, updating from 1.8 to 1.9, you must switch to the **fast-1.9** channel manually.

6. From the **Version** drop-down menu, select the version of the Red Hat Developer Hub Operator that you want to install. The default version is the latest version available in the selected channel.

7. Select the Operator **Installation mode**.

NOTE

The **All namespaces on the cluster (default)**option is selected by default. The **Specific namespace on the cluster**option is not currently supported.

8. In the **Installed Namespace** field, do one of the following actions:

   - Select **Operator recommended Namespace** to create and use the **rhdh-operator** namespace. This option is selected by default.
   - Select **Select a Namespace** to use an alternative namespace.
     - From the **Select Project** drop-down menu, do one of the following actions:
       - Select an existing project.
       - Select **Create Project** to create a new project for the Operator.
         - On the **Create Project** dialog, enter text into the required fields and click **Create**.

IMPORTANT

For enhanced security, better control over the Operator lifecycle, and preventing potential privilege escalation, install the Red Hat Developer Hub Operator in a dedicated default **rhdh-operator** namespace. You can restrict other users' access to the Operator resources through role bindings or cluster role bindings.

You can also install the Operator in another namespace by creating the necessary resources, such as an Operator group. For more information, see Installing global Operators in custom namespaces.

However, if the Red Hat Developer Hub Operator shares a namespace with other Operators, then it shares the same update policy as well, preventing the customization of the update policy. For example, if one Operator is set to manual updates, the Red Hat Developer Hub Operator update policy is also set to manual. For more information, see Colocation of Operators in a namespace.

9. Select the **Update approval** method for the Operator.
    - If you select the **Automatic** option, the Operator is updated without requiring manual confirmation.
    - If you select the **Manual** option, a notification opens when a new update is released in the update channel. The update must be manually approved by an administrator before installation can begin.
10. Click **Install**.

NOTE

If you selected a **Manual** approval strategy, the upgrade status of the subscription remains **Upgrading** until you review and approve the install plan. After you click **Approve** on the **Install Plan** page, the subscription upgrade status changes to **Up to date**.

If you selected an **Automatic** approval strategy, the upgrade status should resolve to **Up to date** without intervention.

Verification

- Immediately after the Operator is installed, the dialog box on the **OperatorHub** page displays the **Installed operator: ready for use** message.
- From the dialog box, do one of the following actions:
    - Click **View Operator** to open the **Operator details** page for the Red Hat Developer Hub Operator.
    - Click **View all installed operators** to open the **Installed Operators** page.

- From the list of installed Operators, locate the Red Hat Developer Hub Operator name and details.
- Click **Red Hat Developer Hub Operator** to open the **Operator details** page for the Red Hat Developer Hub Operator.

CHAPTER 3. PREPARING YOUR EXTERNAL SERVICES

Red Hat Developer Hub relies on external services. Prepare the following required external services:

PostgreSQL database

Developer Hub stores data in a PostgreSQL database. Use an external database for resiliency and include it in your disaster recovery plan.

Redis cache

For efficiency, Developer Hub caches plugin and Techdocs assets when your provide a Redis cache server.

GitHub API access

Provide credentials to a GitHub app to enable access to the GitHub API for repository discovery.

Connection to your identity provider

Provide credentials to your identity provider to enable user provisioning and authentication.

Procedure

- Get your external PostgreSQL database connection strings and certificates.

  postgres-host

  Your PostgreSQL instance Domain Name System (DNS) or IP address.

  postgres-port

  Your PostgreSQL instance port number, such as 5432.

  postres-username

  The user name to connect to your PostgreSQL instance.

  postgres-password

  The password to connect to your PostgreSQL instance.

  postgres-ca.pem, postgres-key.key, postgres-crt.pem

  For security, use TLS certificates to secure the connection to the database.

  1. Get your Redis cache server connection string, such as **rediss://user:pass@cache.example.com:6379**. For security, consider using a **rediss** secure server connection.

  2. Create a GitHub App to allow Developer Hub to access the GitHub API for repository. 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**.

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

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

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

d. Save the following values for the next step:

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

CHAPTER 4. PROVISIONING YOUR CUSTOM RED HAT DEVELOPER HUB CONFIGURATION

To configure Red Hat Developer Hub, provision your custom Red Hat Developer Hub config maps and secrets to {platform-long} before running Red Hat Developer Hub.

TIP

On Red Hat OpenShift Container Platform, you can skip this step to run Developer Hub with the default config map and secret. Your changes on this configuration might get reverted on Developer Hub restart.

Prerequisites

- By using the {platform-cli-link}, you have access, with developer permissions, to the OpenShift cluster aimed at containing your Developer Hub instance.
- You have the connection string to an active Redis server, such as **rediss://user:pass@cache.example.com:6379**. For security, consider using a **rediss** secure server connection. See Chapter 3, *Preparing your external services*.
- You have an external PostgreSQL database, with the following details. See See Chapter 3, *Preparing your external services*.

  postgres-host

  Your PostgreSQL instance Domain Name System (DNS) or IP address.

  postgres-port

  Your PostgreSQL instance port number, such as 5432.

  postres-username

  The user name to connect to your PostgreSQL instance.

  postgres-password

  The password to connect to your PostgreSQL instance.

  postgres-ca.pem, postgres-key.key, postgres-crt.pem

  TLS certificates to secure the connection to the database.

- You have a GitHub App enabling access to the GitHub API for repository discovery, with the following details. See Chapter 3, *Preparing your external services*.

  GITHUB_INTEGRATION_APP_ID

  Your GitHub integration App ID.

  GITHUB_INTEGRATION_CLIENT_ID

  Your GitHub integration App client ID.

  GITHUB_INTEGRATION_CLIENT_SECRET

  Your GitHub integration App client secret.

  GITHUB_INTEGRATION_PRIVATE_KEY_FILE

  Your GitHub integration App private key.

Procedure

1. For security, store your secrets as environment variables values in an OpenShift Container Platform secret, rather than in clear text in your configuration files. Collect all your secrets in the **secrets.txt** file, with one secret per line in **KEY=value** form.

   a. Enter your custom logo.

   ```
   BASE64_EMBEDDED_FULL_LOGO="
   <base64_full_logo_data>"
   BASE64_EMBEDDED_ICON_LOGO="
   <base64_icon_logo_data>"
   ```

   BASE64_EMBEDDED_FULL_LOGO

   Enter your logo for the expanded (pinned) sidebar as a base64 encoded SVG image. To encode your logo in base64, run:

   ```
   $ base64 -i logo.svg
   ```

   BASE64_EMBEDDED_ICON_LOGO

   Enter your logo for the collapsed (unpinned) sidebar as a base64 encoded SVG image.

   b. Enter the connection string to your Redis server that caches plugin assets.

   ```
   REDIS_CONNECTION=rediss://user:pass@cache.example.com:6379
   ```

   c. Enter your GitHub integration credentials:

   ```
   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=github.com
   GITHUB_INTEGRATION_ORGANIZATION=<Enter_your_github_organization_name>
   GITHUB_INTEGRATION_PRIVATE_KEY_FILE= <Enter_the_saved_Private_key>
   ```

   d. Enter your PosgreSQL database secrets:

   ```
   POSTGRES_PASSWORD: <postgres-password>
   POSTGRES_PORT: "<postgres-port>"
   POSTGRES_USER: <postgres-username>
   POSTGRES_HOST: <postgres-host>
   PGSSLMODE: verify-full
   NODE_EXTRA_CA_CERTS: /opt/app-root/src/postgres-crt.pem
   ```

   e. Enter your authentication secrets.

2. Author your custom **app-config.yaml** file. This is the main Developer Hub configuration file. You need a custom **app-config.yaml** file to avoid the Developer Hub installer to revert user edits during upgrades. When your custom **app-config.yaml** file is empty, Developer Hub is using default values.

   a. For a production environment, start with the following setup:

   app-config.yaml

```
app:
  title: <Red Hat Developer Hub>
  branding:
    fullLogo: ${BASE64_EMBEDDED_FULL_LOGO}
    fullLogoWidth: 110px
    iconLogo: ${BASE64_EMBEDDED_ICON_LOGO}
backend:
  cache:
    store: redis
    connection: ${REDIS_CONNECTION}
techdocs:
  cache:
    ttl: 3600000
catalog:
  providers:
    <enter_your_provider_configuration>
integrations:
    <enter_your_integrations_configuration>
permission:
  enabled: true
  rbac:
    admin:
      users:
        - name: user:default/<your_policy_administrator_name>
    pluginsWithPermission:
      - catalog
      - scaffolder
      - permission
```

Most fields use environment variables that you defined in secrets in the previous step.

app

title

Enter your Developer Hub instance display name, such as *<Red Hat Developer Hub>*.

branding

Set your custom logo.
Optionally, customize the width of the branding logo by changing value for the **fullLogoWidth** field. The following units are supported: integer, px, em, rem, percentage.

backend

cache

Enable the plugins assets cache.

techdocs

cache

Enable the Techdocs cache.

catalog

provider

Enter your catalog provider configuration.

integrations

Enter your repository discovery configuration.

permissions

Enable Role-based access control. Enter your policy administrator name.

b. Additionally, provision users and enable authentication with your external identity provider .

3. Author your custom **dynamic-plugins.yaml** file to enable plugins. By default, Developer Hub enables a minimal plugin set, and disables plugins that require configuration or secrets, such as the GitHub repository discovery plugin and the Role-based access control (RBAC) plugin. Enable the GitHub repository discovery and the RBAC features:

dynamic.plugins.yaml

```
includes:
 - dynamic-plugins.default.yaml
plugins:
 - package: ./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github
   disabled: false
 - package: ./dynamic-plugins/dist/backstage-community-plugin-rbac
   disabled: false
```

4. Provision your custom configuration files to your {platform} cluster.

a. Create the *<my-rhdh-project>* {namespace} aimed at containing your Developer Hub instance.

```
$ oc create namespace my-rhdh-project
```

b. Provision your **app-config.yaml** and **dynamic-plugins.yaml** files respectively to the **my-rhdh-app-config**, and **dynamic-plugins-rhdh** config maps in the *<my-rhdh-project>* project.

```
$ oc create configmap my-rhdh-app-config --from-file=app-config.yaml --
namespace=my-rhdh-project
$ oc create configmap dynamic-plugins-rhdh --from-file=dynamic-plugins.yaml --
namespace=my-rhdh-project
```

Alternatively, create the config maps by using the web console .

c. Provision your **secrets.txt** file to the **my-rhdh-secrets** secret in the *<my-rhdh-project>* project.

```
$ oc create secret generic my-rhdh-secrets --from-file=secrets.txt --namespace=my-
rhdh-project
```

Alternatively, create the secret by using the web console .

d. Provision your PosgreSQL TLS certificates to the **my-rhdh-database-secrets** secret in the *<my-rhdh-project>* project.

```
$ oc create secret generic my-rhdh-secrets --from-file=postgres-ca.pem --from-
file=postgres-crt.pem --from-file=postgres-key.key --namespace=my-rhdh-project
```

CHAPTER 5. ENABLING AUTHENTICATION IN RED HAT DEVELOPER HUB (WITH MANDATORY STEPS ONLY)

5.1. UNDERSTANDING AUTHENTICATION AND USER PROVISIONING

Learn about the authentication process from creating user and group entities in the software catalog to user sign-in, and how authentication and catalog plugins enable each step. Understanding this process is essential for successfully configuring your Developer Hub instance, securing access through authorization, and enabling features that rely on synchronized user and group data.

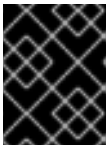
To fully enable catalog features, provision user and group data from the Identity Provider to the Developer Hub software catalog. Catalog provider plugins handle this task asynchronously. These plugins query the Identity Provider (IdP) for relevant user and group information, and create or update corresponding entities in the Developer Hub catalog. Scheduled provisioning ensures that the catalog accurately reflects the users and groups in your organization.

When a user attempts to access Developer Hub, Developer Hub redirects them to a configured authentication provider, such as Red Hat Build of Keycloak (RHBK), GitHub, or Microsoft Azure. This external IdP is responsible for authenticating the user.

On successful authentication, the Developer Hub authentication plugin, configured in your **app-config.yaml** file, processes the response from the IdP, resolves the identity in the Developer Hub software catalog, and establishes a user session within Developer Hub.

Configuring authentication and user provisioning is critical for several reasons.

- Securing your Developer Hub instance by ensuring only authenticated users can gain access.
- Enabling authorization by allowing you to define access controls based on user and group memberships synchronized from your IdP.
- Provisioning user and group data to the catalog is necessary for various catalog features that rely on understanding entity ownership and relationships between users, groups, and software components.



IMPORTANT

Without this provisioning step, features such as displaying who owns a catalog entity might not function correctly.

TIP

To explore Developer Hub features in a non-production environment, you can:

- To use Developer Hub without external IdP, enable the guest user to skip configuring authentication and authorization, log in as the guest user, and access all Developer Hub features.
- To use Developer Hub without authorization policies and features relying on the software catalog, you can enable the **dangerouslyAllowSignInWithoutUserInCatalog** resolver option. This setting bypasses the check requiring a user to be in the catalog but still enforces authentication.

IMPORTANT

Developer Hub uses a one-way synchronization model, where user and group data flow from your Identity Provider to the Developer Hub software catalog. As a result, deleting users or groups manually through the Developer Hub Web UI or REST API might be ineffective or cause inconsistencies, since Developer Hub will create those entities again during the next import.

5.2. AUTHENTICATING WITH THE GUEST USER

For trial or non-production environments, you can enable guest access to skip configuring authentication and authorization and explore Developer Hub features.

5.2.1. Authenticating with the Guest user on an Operator-based installation

For trial or non-production environments installed by using the Red Hat Developer Hub Operator, you can enable guest access to skip configuring authentication and authorization and explore Developer Hub features.

Prerequisites

- You added a custom Developer Hub application configuration , and have enough permissions to change it.
- You use the Red Hat Developer Hub Operator to run Developer Hub .

Procedure

- Add the following content to the **app-config.yaml** file:

```
auth:
  environment: development
  providers:
    guest:
      dangerouslyAllowOutsideDevelopment: true
```

Verification

1. Go to the Developer Hub login page.
2. To log in with the Guest user account, click **Enter** in the **Guest** tile.
3. In the Developer Hub **Settings** page, your profile name is **Guest**.
4. You can use Developer Hub features.

5.2.2. Authenticating with the Guest user on a Helm-based installation

For trial or non-production environments installed by using the Red Hat Developer Hub Helm chart, you can enable guest access to skip configuring authentication and authorization and explore Developer Hub features.

Prerequisites

- You added a custom Developer Hub application configuration , and have enough permissions to change it.
- You use the Red Hat Developer Hub Helm chart to run Developer Hub .

Procedure

- Add following content to your Red Hat Developer Hub Helm Chart:

```
upstream:
  backstage:
    appConfig:
      app:
        baseUrl: 'https://{{- include "janus-idp.hostname" . }}'
      auth:
        environment: development
        providers:
          guest:
            dangerouslyAllowOutsideDevelopment: true
```

Verification

1. Go to the Developer Hub login page.
2. To log in with the Guest user account, click **Enter** in the **Guest** tile.
3. In the Developer Hub **Settings** page, your profile name is **Guest**.
4. You can use Developer Hub features.

5.3. ENABLING USER AUTHENTICATION WITH RED HAT BUILD OF KEYCLOAK (RHBK)

Authenticate users with Red Hat Build of Keycloak (RHBK), by provisioning the users and groups from RHBK to the Developer Hub software catalog, and configuring the OpenID Connect (OIDC) authentication provider in Red Hat Developer Hub.

Prerequisites

- You added a custom Developer Hub application configuration , and have enough permissions to change it.
- You have enough permissions in RHSSO to create and manage a realm and a client.

  TIP

  Alternatively, ask your RHBK administrator to prepare in RHBK the required realm and client.

Procedure

1. Register your Developer Hub app in RHBK:
   a. Use an existing realm, or create a realm, with a distinctive **Name** such as *<my_realm>*. Save the value for the next step:

- **RHBK realm base URL**, such as: *<your_rhbk_URL>*/realms/*<your_realm>*.

b. To register your Developer Hub in RHBK, in the created realm, secure the first application, with:

i. **Client ID**: A distinctive client ID, such as *<RHDH>*.

ii. **Valid redirect URIs**: Set to the OIDC handler URL: **https://*<my_developer_hub_domain>*/api/auth/oidc/handler/frame**.

iii. Go to the **Credentials** tab and copy the **Client secret**.

iv. Save the values for the next step:

- **Client ID**
- **Client Secret**

c. To prepare for the verification steps, in the same realm, get the credential information for an existing user or create a user. Save the user credential information for the verification steps.

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

KEYCLOAK_CLIENT_ID

Enter the saved **Client ID**.

KEYCLOAK_CLIENT_SECRET

Enter the saved **Client Secret**.

KEYCLOAK_BASE_URL

Enter the saved **RHBK realm base URL**

KEYCLOAK_REALM

Enter the realm name to provision users.

KEYCLOAK_LOGIN_REALM

Enter the realm name to authenticate users.

3. Enable the Keycloak catalog provider plugin in your **dynamic-plugins.yaml** file.
The plugin is named after RHBK upstream project.

This plugin imports RHBK users and groups to the Developer Hub software catalog.

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

4. Enable provisioning RHBK users and groups to the Developer Hub software catalog, by adding the **catalog.providers.keycloakOrg** section to your **app-config.yaml** file:

```
catalog:
  providers:
    keycloakOrg:
```

```
      default:
        baseUrl: ${KEYCLOAK_BASE_URL}
        clientId: ${KEYCLOAK_CLIENT_ID}
        clientSecret: ${KEYCLOAK_CLIENT_SECRET}
        realm: ${KEYCLOAK_REALM}
        loginRealm: ${KEYCLOAK_LOGIN_REALM}
```

baseUrl

Enter your RHBK server URL, defined earlier.

clientId

Enter your Developer Hub application client ID in RHBK, defined earlier.

clientSecret

Enter your Developer Hub application client secret in RHBK, defined earlier.

realm

Enter the realm name to provision users.

loginRealm

Enter the realm name to authenticate users.
Optional: To configure optional fields, see Configuring Red Hat Developer Hub .

5. Enable the RHBK authentication provider, by adding the OIDC provider section in your **app-config.yaml** file:

```
auth:
  environment: production
  providers:
    oidc:
      production:
        metadataUrl: ${KEYCLOAK_BASE_URL}
        clientId: ${KEYCLOAK_CLIENT_ID}
        clientSecret: ${KEYCLOAK_CLIENT_SECRET}
        prompt: auto
signInPage: oidc
```

environment: production

Mark the environment as **production** to hide the Guest login in the Developer Hub home page.

metadataUrl, clientId, clientSecret

Configure the OIDC provider with your secrets.

prompt

Enter **auto** to allow the identity provider to automatically determine whether to prompt for credentials or bypass the login redirect if an active RHSSO session exists.
The identity provider defaults to **none**, which assumes that you are already logged in. Sign-in requests without an active session are rejected.

signInPage

Enter **oidc** to enable the OIDC provider as default sign-in provider.
Optional: To configure optional fields, see Configuring Red Hat Developer Hub .

Verification

1. To verify user and group provisioning, check the console logs.
   Successful synchronization example:

   ```
   2025-06-27T16:02:34.647Z catalog info Read 5 Keycloak users and 3 Keycloak groups in 0.4 seconds. Committing... class="KeycloakOrgEntityProvider" taskId="KeycloakOrgEntityProvider:default:refresh" taskInstanceId="db55c34b-46b3-402b-b12f-2fbc48498e82" trace_id="606f80a9ce00d1c86800718c4522f7c6" span_id="7ebc2a254a546e90" trace_flags="01"

   2025-06-27T16:02:34.650Z catalog info Committed 5 Keycloak users and 3 Keycloak groups in 0.0 seconds. class="KeycloakOrgEntityProvider" taskId="KeycloakOrgEntityProvider:default:refresh" taskInstanceId="db55c34b-46b3-402b-b12f-2fbc48498e82" trace_id="606f80a9ce00d1c86800718c4522f7c6" span_id="7ebc2a254a546e90" trace_flags="01"
   ```

2. To verify RHBK user authentication:

   a. Go to the Developer Hub login page.

   b. Your Developer Hub sign-in page displays **Sign in using OIDC** and the Guest user sign-in is disabled.

   c. Log in with OIDC by using the saved **Username** and **Password** values.

5.4. ENABLING USER AUTHENTICATION WITH GITHUB

Authenticate users with GitHub by provisioning the users and groups from GitHub to the Developer Hub software catalog, and configuring the GitHub authentication provider in Red Hat Developer Hub.

Prerequisites

- You have enough permissions in GitHub to create and manage a GitHub App.

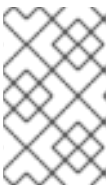
  TIP

  Alternatively, ask your GitHub administrator to prepare the required GitHub App.

- You have added a custom Developer Hub application configuration , and have enough permissions to change it.

Procedure

1. Allow Developer Hub to authenticate with GitHub, by creating a GitHub App.

   NOTE

   Use a GitHub App instead of an OAuth app to use fine-grained permissions, use short-lived tokens, scale with the number of installations by avoiding rate limits, and have a more transparent integration by avoiding to request user input.

   a. Register a GitHub App with the following configuration:

GitHub App name

Enter a unique name identifying your GitHub App, such as **authenticating-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".

Organization permissions

Enable **Read-only** access to **Members**.

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

d. Save the following values for the next step:

- **Client ID**
- **Client secret**

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

GITHUB_CLIENT_ID

Enter the saved **Client ID**.

GITHUB_CLIENT_SECRET

Enter the saved **Client Secret**.

GITHUB_URL

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

GITHUB_ORG

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

3. Enable the GitHub catalog provider plugin in your **dynamic-plugins.yaml** file to import GitHub users and groups to the Developer Hub software catalog.

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

4. Enable provisioning GitHub users and groups to the Developer Hub software catalog by adding the GitHub catalog provider section to your **app-config.yaml** file:

```
catalog:
  providers:
    githubOrg:
      id: githuborg
      githubUrl: "${GITHUB_URL}"
      orgs: [ "${GITHUB_ORG}" ]
      schedule:
        frequency:
          minutes: 30
        initialDelay:
          seconds: 15
        timeout:
          minutes: 15
```

id

Enter a stable identifier for this provider, such as **githuborg**.

Entities from this provider are associated with this identifier. Therefore, do not to change the identifier over time since that might lead to orphaned entities or conflicts.

githubUrl

Enter the configured secret variable name: **${GITHUB_URL}**.

orgs

Enter the configured secret variable name: **${GITHUB_ORG}**.

schedule.frequency

Enter your schedule frequency, in the cron, ISO duration, or "human duration" format.

schedule.timeout

Enter your schedule timeout, in the ISO duration or "human duration" format.

schedule.initialDelay

Enter your schedule initial delay, in the ISO duration or "human duration" format.

1. Enable the GitHub authentication provider, by adding the GitHub authentication provider section to your **app-config.yaml** file:

```
auth:
  environment: production
  providers:
    github:
      production:
        clientId: ${GITHUB_CLIENT_ID}
        clientSecret: ${GITHUB_CLIENT_SECRET}
signInPage: github
```

environment

Enter **production** to disable the Guest login option in the Developer Hub login page.

clientId

Enter the configured secret variable name: **${GITHUB_CLIENT_ID}**.

clientSecret

Enter the configured secret variable name: **${GITHUB_CLIENT_SECRET}**.

signInPage

Enter **github** to enable the GitHub provider as your Developer Hub sign-in provider.
Optional: To configure optional fields, see Configuring Red Hat Developer Hub .

Verification

1. Verify user and group provisioning by checking the console logs.
Successful synchronization example:


```
{"class":"GithubMultiOrgEntityProvider","level":"info","message":"Reading GitHub users and teams for org: rhdh-dast","plugin":"catalog","service":"backstage","target":"https://github.com","taskId":"GithubMultiOrgEntityProvider:githuborg:refresh","taskInstanceId":"801b3c6c-167f-473b-b43e-e0b4b780c384","timestamp":"2024-09-09 23:55:58"}
{"class":"GithubMultiOrgEntityProvider","level":"info","message":"Read 7 GitHub users and 2 GitHub groups in 0.4 seconds. Committing...","plugin":"catalog","service":"backstage","target":"https://github.com","taskId":"GithubMultiOrgEntityProvider:githuborg:refresh","taskInstanceId":"801b3c6c-167f-473b-b43e-e0b4b780c384","timestamp":"2024-09-09 23:55:59"}
```


2. To verify GitHub authentication:
   a. Go to the Developer Hub login page.
   b. Your Developer Hub sign-in page displays **Sign in using GitHub** and the Guest user sign-in is disabled.
   c. Log in with a GitHub account.

Additional resources

- Integrating Red Hat Developer Hub with GitHub

5.5. ENABLING USER AUTHENTICATION WITH MICROSOFT AZURE

Authenticate users with Microsoft Azure by provisioning the users and groups from Azure to the Developer Hub software catalog, and configuring the Azure authentication provider in Red Hat Developer Hub.

Prerequisites

- You have the permission to register an application in Azure.

  TIP

  Alternatively, ask your Azure administrator to prepare the required Azure application.

- You added a custom Developer Hub application configuration , and have enough permissions to change it.
- Your Developer Hub backend can access the following hosts:

  login.microsoftonline.com

The Microsoft Azure authorization server, which enables the authentication flow.

graph.microsoft.com

The server for retrieving organization data, including user and group data, to import into the Developer Hub catalog.

Procedure

1. Register your Developer Hub app in Azure, by using the Azure portal.

   a. Sign in to the Microsoft Entra admin center.

   b. Optional: If you have access to multiple tenants, use the **Settings** icon in the top menu to switch to the tenant in which you want to register the application from the **Directories + subscriptions** menu.

   c. Browse to **Applications > App registrations**, and create a **New registration** with the configuration:

      Name

      Enter a name to identify your application in Azure, such as *<Authenticating with Developer Hub>*.

      Supported account types

      Select **Accounts in this organizational directory only**

      Redirect URI

      Select a platform

      Select **Web**.

      URL

      Enter the backend authentication URI set in Developer Hub:
      https://*<my_developer_hub_domain>*/api/auth/microsoft/handler/frame

   d. On the **Applications > App registrations >*<Authenticating with Developer Hub>*> Manage > API permissions** page, **Add a Permission**, **Microsoft Graph**, select the following permissions:

      Application Permissions

      GroupMember.Read.All, User.Read.All

      Enter permissions that enable provisioning user and groups to the Developer Hub software catalog.
      Optional: **Grant admin consent** for these permissions. Even if your company does not require admin consent, consider doing so as it means users do not need to individually consent the first time they access Developer Hub.

      Delegated Permissions

      User.Read, email, offline_access, openid, profile

      Enter permissions that enable authenticating users.
      Optional: Enter optional custom scopes for the Microsoft Graph API that you define both here and in your **app-config.yaml** Developer Hub configuration file.

e. On the **Applications > App registrations >*<Authenticating with Developer Hub>*> Manage > Certificates & secrets** page, in the **Client secrets** tab, create a **New client secret**.

f. Save the following values for the next step:

- **Directory (tenant) ID**
- **Application (client) ID**
- **Application (client) Secret ID**

2. Add your Azure credentials to Developer Hub, by adding the following key/value pairs to your Developer Hub secrets:

MICROSOFT_TENANT_ID

Enter your saved **Directory (tenant) ID**.

MICROSOFT_CLIENT_ID

Enter your saved **Application (client) ID**.

MICROSOFT_CLIENT_SECRET

Enter your saved **Application (client) secret**.

3. Enable the Microsoft Graph catalog provider plugin in your **dynamic-plugins.yaml** file. This plugin imports Azure users and groups to the Developer Hub software catalog.

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

4. Enable provisioning Azure users and groups to the Developer Hub software catalog, by adding the Microsoft Graph catalog provider section in your **app-config.yaml** file:

```
catalog:
  providers:
    microsoftGraphOrg:
      providerId:
        target: https://graph.microsoft.com/v1.0
        tenantId: ${MICROSOFT_TENANT_ID}
        clientId: ${MICROSOFT_CLIENT_ID}
        clientSecret: ${MICROSOFT_CLIENT_SECRET}
        schedule:
          frequency:
            hours: 1
          timeout:
            minutes: 50
          initialDelay:
            minutes: 50
```

target

Enter **https://graph.microsoft.com/v1.0** to define the MSGraph API endpoint the provider is connecting to. You might change this parameter to use a different version, such as the beta endpoint.

tenandId

Enter the configured secret variable name: **${MICROSOFT_TENANT_ID}**.

clientId

Enter the configured secret variable name: **${MICROSOFT_CLIENT_ID}**.

clientSecret

Enter the configured secret variable name: **${MICROSOFT_CLIENT_SECRET}**.

schedule

frequency

Enter the schedule frequency in the cron, ISO duration, or human duration format. In a large organization, user provisioning might take a long time, therefore avoid using a low value.

timeout

Enter the schedule timeout in the ISO duration or human duration format. In a large organization, user provisioning might take a long time, therefore avoid using a low value.

initialDelay

Enter the schedule initial delay in the ISO duration or human duration format.
Optional: To configure optional fields, see Configuring Red Hat Developer Hub .

Enable Azure authentication, by adding the Microsoft authentication provider to your **app-config.yaml** file content:

```
auth:
  environment: production
  providers:
    microsoft:
      production:
        clientId: ${MICROSOFT_CLIENT_ID}
        clientSecret: ${MICROSOFT_CLIENT_SECRET}
        tenantId: ${MICROSOFT_TENANT_ID}
signInPage: microsoft
```

environment

Enter **production** to disable the **Guest** login option in the Developer Hub login page.

clientId

Enter the configured secret variable name: **${MICROSOFT_CLIENT_ID}**.

clientSecret

Enter the configured secret variable name: **${MICROSOFT_CLIENT_SECRET}**.

tenantId

Enter the configured secret variable name: **${MICROSOFT_TENANT_ID}**.

signInPage

Enter **microsoft** to set the Azure provider as your Developer Hub sign-in provider.
Optional: To configure optional fields, see Configuring Red Hat Developer Hub .

Verification

1. To verify user and group provisioning, check the console logs for **MicrosoftGraphOrgEntityProvider** events.
   Successful synchronization example:

   > 2025-06-23T13:37:55.804Z catalog info Read 9 msgraph users and 3 msgraph groups in 1.5 seconds. Committing... class="MicrosoftGraphOrgEntityProvider" taskId="MicrosoftGraphOrgEntityProvider:providerId:refresh" taskInstanceId="e104a116-6481-4ceb-9bc4-0f8f9581f959" trace_id="e4c633659cffd6b1529afa55a5bfbad7" span_id="76affd0420e8baa6" trace_flags="01"
   >
   > 2025-06-23T13:37:55.811Z catalog info Committed 9 msgraph users and 3 msgraph groups in 0.0 seconds. class="MicrosoftGraphOrgEntityProvider" taskId="MicrosoftGraphOrgEntityProvider:providerId:refresh" taskInstanceId="e104a116-6481-4ceb-9bc4-0f8f9581f959" trace_id="e4c633659cffd6b1529afa55a5bfbad7" span_id="76affd0420e8baa6" trace_flags="01"

2. To verify Azure user authentication:

   a. Go to the Developer Hub login page.

   b. Your Developer Hub sign-in page displays **Sign in using Microsoft** and the Guest user sign-in is disabled.

   c. Log in with an Azure account.

CHAPTER 6. USING THE RED HAT DEVELOPER HUB OPERATOR TO RUN DEVELOPER HUB WITH YOUR CUSTOM CONFIGURATION

Use the Developer Hub Operator to run Red Hat Developer Hub with your custom configuration by creating your Backstage custom resource (CR) that can perform the following actions:

- Mount files provisioned in your custom config maps.
- Inject environment variables provisioned in your custom secrets.

Prerequisites

- By using the OpenShift CLI (**oc**), you have access, with developer permissions, to the OpenShift Container Platform cluster aimed at containing your Developer Hub instance.
- Chapter 2, *Installing the Red Hat Developer Hub Operator*
- Chapter 4, *Provisioning your custom Red Hat Developer Hub configuration*

Procedure

1. Author your Backstage CR in a **my-rhdh-custom-resource.yaml** file to use your custom config maps and secrets.

   my-rhdh-custom-resource.yaml custom resource example with dynamic plugins and RBAC policies config maps, and external PostgreSQL database secrets.

```
apiVersion: rhdh.redhat.com/v1alpha3
kind: Backstage
metadata:
  name: <my-rhdh-custom-resource>
spec:
  application:
    appConfig:
      mountPath: /opt/app-root/src
      configMaps:
        - name: my-rhdh-app-config
        - name: rbac-policies
    dynamicPluginsConfigMapName: dynamic-plugins-rhdh
    extraEnvs:
      envs:
        - name: HTTP_PROXY
          value: 'http://10.10.10.105:3128'
        - name: HTTPS_PROXY
          value: 'http://10.10.10.106:3128'
        - name: NO_PROXY
          value: 'localhost,example.org'
      secrets:
        - name: my-rhdh-secrets
    extraFiles:
      mountPath: /opt/app-root/src
      secrets:
        - name: my-rhdh-database-certificates-secrets
```

```
        key: postgres-crt.pem, postgres-ca.pem, postgres-key.key
    replicas: 2
  database:
    enableLocalDb: false
```

application

appConfig

Register your **my-rhdh-app-config** and **rbac-policies** config maps.

dynamicPluginsConfigMapName

Register your **dynamic-plugins-rhdh** config map.

extraEnvs

env

Enter your proxy environment variables.

secrets

Register your **<my_product_secrets>** and **my-rhdh-database-secrets** secrets.

extraFiles

secrets

Register the **postgres-crt.pem**, **postgres-ca.pem**, and **postgres-key.key** files contained in the **my-rhdh-database-certificates-secrets** secret.

replicas

Enable high availability (HA) by increasing the replicas count to a value higher or equal to 2.

database

enableLocalDb

Use your external PostgreSQL database rather than the internal PostgreSQL database.

2. Apply your Backstage CR to start or update your Developer Hub instance.

```
$ oc apply --filename=my-rhdh-custom-resource.yaml --namespace=my-rhdh-project
```

CHAPTER 7. CUSTOMIZING THE THEME MODE FOR YOUR DEVELOPER HUB INSTANCE

You can choose one of the following theme modes for your Developer Hub instance:

- **Light**
- **Dark**
- **Auto**

NOTE

In Developer Hub, theme configurations are used to change the look and feel of different UI components. So, you might notice changes in different UI components, such as buttons, tabs, sidebars, cards, and tables along with some changes in background color and font used on the RHDH pages.

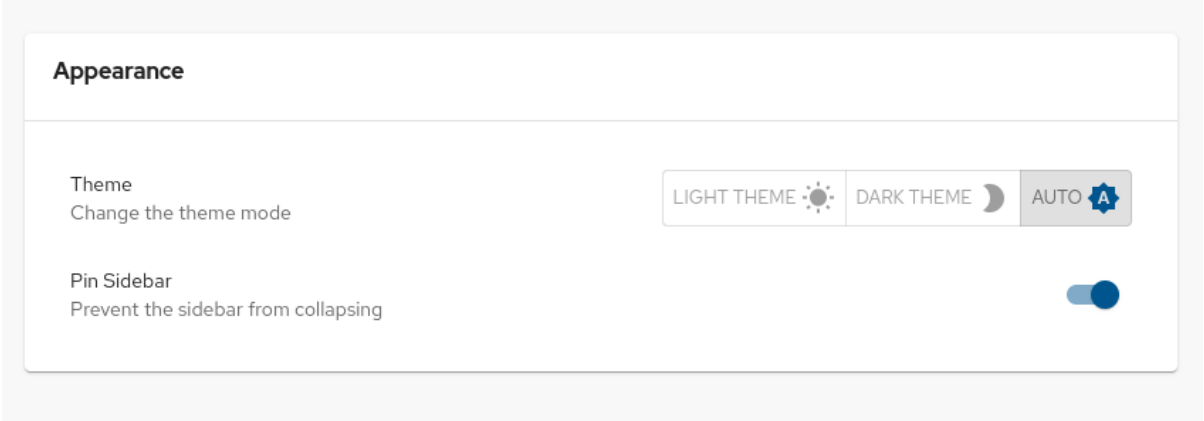
The default theme mode is **Auto**, which automatically sets the light or dark theme based on your system preferences.

Prerequisites

- You are logged in to the Developer Hub web console.

Procedure

1. From the Developer Hub web console, click **Settings**.
2. From the **Appearance** panel, click **Light**, **Dark**, or **Auto** to change the theme mode.



CHAPTER 8. MANAGING ROLE-BASED ACCESS CONTROLS (RBAC) USING THE RED HAT DEVELOPER HUB WEB UI

Policy administrators can use the Developer Hub web interface (Web UI) to allocate specific roles and permissions to individual users or groups. Allocating roles ensures that access to resources and functionalities is regulated across the Developer Hub.

With the policy administrator role in Developer Hub, you can assign permissions to users and groups. This role allows you to view, create, modify, and delete the roles using Developer Hub Web UI.

8.1. CREATING A ROLE IN THE RED HAT DEVELOPER HUB WEB UI

You can create a role in the Red Hat Developer Hub using the Web UI.

Prerequisites

- You have enabled RBAC, have a policy administrator role in Developer Hub, and have added plugins with permission.

Procedure

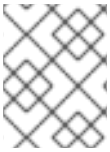
1. Go to **Administration** at the bottom of the sidebar in the Developer Hub.
   The **RBAC** tab appears, displaying all the created roles in the Developer Hub.
2. (Optional) Click any role to view the role information on the **OVERVIEW** page.
3. Click **CREATE** to create a role.
4. Enter the name and description of the role in the given fields and click **NEXT**.
5. Add users and groups using the search field, and click **NEXT**.
6. Select **Plugin** and **Permission** from the drop-downs in the **Add permission policies** section.
7. Select or clear the **Policy** that you want to set in the **Add permission policies** section, and click **NEXT**.
8. Review the added information in the **Review and create** section.
9. Click **CREATE**.

Verification

The created role appears in the list available in the **RBAC** tab.

8.2. EDITING A ROLE IN THE RED HAT DEVELOPER HUB WEB UI

You can edit a role in the Red Hat Developer Hub using the Web UI.



NOTE

The policies generated from a **policy.csv** or ConfigMap file cannot be edited or deleted using the Developer Hub Web UI.

Prerequisites

- You have enabled RBAC, have a policy administrator role in Developer Hub, and have added plugins with permission.
- The role that you want to edit is created in the Developer Hub.

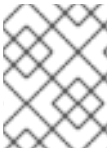
Procedure

1. Go to **Administration** at the bottom of the sidebar in the Developer Hub.
   The **RBAC** tab appears, displaying all the created roles in the Developer Hub.
2. (Optional) Click any role to view the role information on the **OVERVIEW** page.
3. Select the edit icon for the role that you want to edit.
4. Edit the details of the role, such as name, description, users and groups, and permission policies, and click **NEXT**.
5. Review the edited details of the role and click **SAVE**.

After editing a role, you can view the edited details of a role on the **OVERVIEW** page of a role. You can also edit a role's users and groups or permissions by using the edit icon on the respective cards on the **OVERVIEW** page.

8.3. DELETING A ROLE IN THE RED HAT DEVELOPER HUB WEB UI

You can delete a role in the Red Hat Developer Hub using the Web UI.



NOTE

The policies generated from a **policy.csv** or ConfigMap file cannot be edited or deleted using the Developer Hub Web UI.

Prerequisites

- You have enabled RBAC and have a policy administrator role in Developer Hub .
- The role that you want to delete is created in the Developer Hub.

Procedure

1. Go to **Administration** at the bottom of the sidebar in the Developer Hub.
   The **RBAC** tab appears, displaying all the created roles in the Developer Hub.
2. (Optional) Click any role to view the role information on the **OVERVIEW** page.
3. Select the delete icon from the **Actions** column for the role that you want to delete.
   Delete this role? pop-up appears on the screen.
4. Click **DELETE**.