Red Hat Developer Hub 1.1

Getting started with Red Hat Developer Hub

Last Updated: 2024-06-25

Legal Notice

Abstract

This document walks you through the requirements and instructions to install and configure the Red Hat Developer Hub.

Table of Contents

PREFACE

As a developer, you can use Red Hat Developer Hub to experience a streamlined development environment. Red Hat Developer Hub is driven by a centralized software catalog, providing efficiency to your microservices and infrastructure. It enables your product team to deliver quality code without any compromises.

RED HAT DEVELOPER HUB SUPPORT

If you experience difficulty with a procedure described in this documentation, visit the Red Hat Customer Portal. You can use the Red Hat Customer Portal for the following purposes:

- To search or browse through the Red Hat Knowledgebase of technical support articles about Red Hat products.
- To create a support case for Red Hat Global Support Services (GSS). For support case creation, select **Red Hat Developer Hub**as the product and select the appropriate product version.

CHAPTER 1. OVERVIEW OF RED HAT DEVELOPER HUB

Red Hat Developer Hub (Developer Hub) serves as an open developer platform designed for building developer portals. Using Developer Hub, the engineering teams can access a unified platform that streamlines the development process and provides a variety of tools and resources to build high-quality software efficiently.

The goal of Developer Hub is to address the difficulties associated with creating and sustaining developer portals using:

- A centralized dashboard to view all available developer tools and resources to increase productivity
- Self-service capabilities, along with guardrails, for cloud-native application development that complies with enterprise-class best practices
- Proper security and governance for all developers across the enterprise

The Red Hat Developer Hub simplifies decision-making by providing a developer experience that presents a selection of internally approved tools, programming languages, and various developer resources within a self-managed portal. This approach contributes to the acceleration of application development and the maintenance of code quality, all while fostering innovation.

CHAPTER 2. INSTALLING RED HAT DEVELOPER HUB

Administrative users can configure roles, permissions, and other settings to enable other authorized users to install Red Hat Developer Hub on multiple platforms. You can install Developer Hub with a Helm chart or with the Red Hat Developer Hub Operator.

For more information about installing Developer Hub, see the Red Hat Developer Hub Administration guide.

CHAPTER 3. SUPPORTED CONFIGURATIONS FOR RED HAT DEVELOPER HUB

This section describes the configurations that are required to access the Red Hat Developer Hub, including:

- Custom applications configuration
- Source control configuration for Developer Hub Catalog

3.1. ADDING A CUSTOM APPLICATION CONFIGURATION FILE TO RED HAT OPENSHIFT CONTAINER PLATFORM

To access the Red Hat Developer Hub, you must add a custom application configuration file to OpenShift. In OpenShift Container Platform, you can use the following content as a base template to create a ConfigMap named **app-config-rhdh**:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app-config-rhdh
data:
  app-config-rhdh.yaml: |
    app:
      title: Red Hat Developer Hub
```

Prerequisites

- You have created an Red Hat OpenShift Container Platform account.

Procedure

1. From the OpenShift Container Platform web console, select the **ConfigMaps** tab.
2. Click **Create ConfigMap**.
3. From **Create ConfigMap** page, select the **YAML view** option in **Configure via** and make the changes to the file, if needed.
4. Click **Create**.
5. Go to the **Helm** tab.
   The list of Helm Releases appears on the page.
6. Click the overflow menu on a Helm release and select **Upgrade**.
7. Use one of the following views to edit the Helm configuration:
   - Using **Form view**
     a. Expand **Root Schema → Backstage chart schema → Backstage parameters → Extra app configuration files to inline into command arguments**.
     b. Click the **Add Extra app configuration files to inline into command arguments** link.

c. Enter the value in the following fields:

- configMapRef: **app-config-rhdh**
- filename: **app-config-rhdh.yaml**

d. Click **Upgrade**.

- Using **YAML view**

a. Set the value of the **upstream.backstage.extraAppConfig.configMapRef** and **upstream.backstage.extraAppConfig.filename** parameters in the following manner:

```
# ... other Red Hat Developer Hub Helm Chart configurations
upstream:
  backstage:
    extraAppConfig:
      - configMapRef: app-config-rhdh
        filename: app-config-rhdh.yaml
# ... other Red Hat Developer Hub Helm Chart configurations
```

b. Click **Upgrade**.

3.2. ADDING SOURCE CONTROL FOR CATALOG IN RED HAT DEVELOPER HUB

To populate the Catalog in Red Hat Developer Hub, you need to add software templates, and to add the templates, you must enable a source control such as GitHub, GitLab, or BitBucket.

Prerequisites

- You have a GitHub account.
- You have an account on the Red Hat OpenShift cluster.
- You have installed the Developer Hub, otherwise the GitHub login fails. For more information about installation, see Chapter 2, *Installing Red Hat Developer Hub* .

3.2.1. Configuring GitHub authentication

The configuration of GitHub authentication is required to enable the GitHub OAuth login in Developer Hub.

Procedure

1. In the Red Hat OpenShift cluster, navigate to the main page of the GitHub organization where you want to create the OAuth application.
2. Click **Settings → Developer Settings → OAuth Apps → Register an application**
3. Enter the application name as **Developer Hub**.
4. Add the following URL as the **Homepage URL**: **https://developer-hub-<NAMESPACE_NAME>.<OPENSHIFT_ROUTE_HOST>/**

5. Add the following URL as **Authorization callback URL**: **https://developer-hub-<NAMESPACE_NAME>.<OPENSHIFT_ROUTE_HOST>/api/auth/github/handler/frame**

6. Clear the **Enable Device Flow** checkbox.

7. Click **Register application** to create your OAuth application.

8. After creating the application, click **Generate a new client secret** and copy the generated client secret.

9. In OpenShift, click **ConfigMaps**.

10. Generate a key/value secret named 'github-secrets' using the provided environment variables as keys, and input the values you generated for your GitHub OAuth application:

    a. In Red Hat OpenShift, go to the **Secrets** tab and click **Create**.

    b. Select **Key/value secret**

    c. Enter **Secret name** as **github-secrets**.

    d. Add environment variables as **Key** and **Value** and click **Create**.

    Table 3.1. Environment variables

Key	Value
GITHUB_OAUTH_CLIENT_ID	Client ID from OAuth application
GITHUB_OAUTH_CLIENT_SECRET	Client Secret from OAuth application

11. Modify your **app-config-rhdh** ConfigMap to include the GitHub authentication configuration as follows:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app-config-rhdh
data:
  app-config-rhdh.yaml: |
    app:
      title: Red Hat Developer Hub
    auth:
      # see https://backstage.io/docs/auth/ to learn about auth providers
      environment: development
      providers:
        github:
          development:
            clientId: ${GITHUB_OAUTH_CLIENT_ID}
            clientSecret: ${GITHUB_OAUTH_CLIENT_SECRET}
```

12. Click **Save**.

13. Navigate to the **Helm** tab and select **Upgrade**.

14. Use one of the following views to edit the Helm configuration:

    - Using **Form view**

        a. Expand **Root Schema → Backstage Chart Schema → Backstage Parameters → Backstage container environment variables from existing Secrets**.

        b. Click the **Add Backstage container environment variables from existing Secrets** link.

        c. Enter **github-secrets** as the value.

        d. Click **Upgrade**.

    - Using **YAML view**

        a. Set the value of the **upstream.backstage.extraEnvVarsSecrets** parameter to **github-secrets** as shown in the following example:

```
# other Red Hat Developer Hub Helm Chart configurations
upstream:
  backstage:
    # other Red Hat Developer Hub Helm Chart configurations
    extraEnvVarsSecrets:
      - github-secrets
    # other Red Hat Developer Hub Helm Chart configurations
```

        b. Click **Upgrade**.

3.2.2. Configuring GitHub integration

The configuration of GitHub is required to enable the GitHub plugins in Developer Hub.

Procedure

1. In the Red Hat OpenShift cluster, navigate to the main page of the GitHub organization where you want to create the OAuth application.
2. Click **Settings → Developer Settings → GitHub Apps → New GitHub App**.
3. Enter the application name as **Developer Hub**.
4. Add the following URL as the **Homepage URL**:
   https://developer-hub-<NAMESPACE_NAME>.<OPENSHIFT_ROUTE_HOST>/
5. Add the following URL as **Authorization callback URL**:
   https://developer-hub-<NAMESPACE_NAME>.<OPENSHIFT_ROUTE_HOST>/api/auth/github/handler/frame
6. Deselect **Webhook URL → Active**.
7. Under the **Where can this GitHub App be installed?** section, ensure that **Only on this account** is selected.
8. Click **Register application**.

9. After creating the application, click **Generate a new client secret** and copy the generated client secret.

10. Click **Generate a private key** at the bottom of the page and download the generated file.

11. In OpenShift, click **ConfigMaps**.

12. Generate a key/value secret named 'github-secrets' using the provided environment variables as keys, and input the values you generated for your GitHub OAuth application:

    a. In Red Hat OpenShift, go to the **Secrets** tab and click **Create**.

    b. Select **Key/value secret**

    c. Enter **Secret name** as **github-secrets**.

    d. Add environment variables as **Key** and **Value** and click **Create**.

Table 3.2. Environment variables

Key	Value
GITHUB_APP_APP_ID	App ID from GitHub application
GITHUB_APP_CLIENT_ID	Client ID from GitHub application
GITHUB_APP_CLIENT_SECRET	Client Secret from GitHub application
GITHUB_APP_WEBHOOK_URL	Enter "none"
GITHUB_APP_WEBHOOK_SECRET	Enter "none"
GITHUB_APP_PRIVATE_KEY	Upload the private key that was downloaded

13. Modify your **app-config-rhdh** ConfigMap to include the GitHub integration configuration as follows:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app-config-rhdh
data:
  app-config-rhdh.yaml: |
    app:
      title: Red Hat Developer Hub
    integrations:
      github:
        - host: github.com
          apps:
            - appId: ${GITHUB_APP_APP_ID}
              clientId: ${GITHUB_APP_CLIENT_ID}
              clientSecret: ${GITHUB_APP_CLIENT_SECRET}
              webhookUrl: ${GITHUB_APP_WEBHOOK_URL}
```

```
webhookSecret: ${GITHUB_APP_WEBHOOK_SECRET}
privateKey: |
  ${GITHUB_APP_PRIVATE_KEY}
```

14. Click **Toplogy → developer hub → Actions** (drop-down) **→ Restart rollout**.

3.2.3. Enabling GitHub discovery in Red Hat Developer Hub

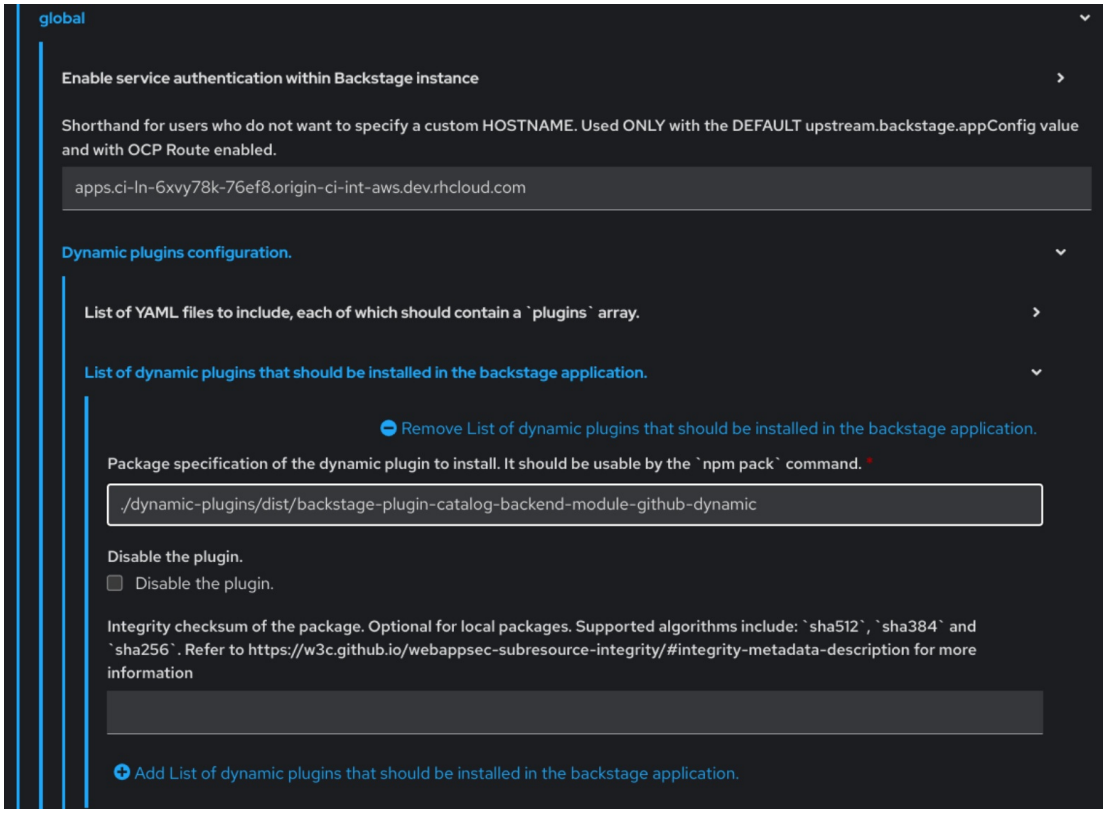
You can enable GitHub discoverability for your components in Developer Hub, such as any repositories that contain **catalog-info.yaml** file.

Prerequisites

- You have set up the GitHub integration. For more information, see Section 3.2.2, "Configuring GitHub integration".

Procedure

1. In the **Developer** perspective of the OpenShift Container Platform web console, go to the **Helm** tab.
2. Click the overflow menu on a Helm release and select **Upgrade**.
3. Use one of the following views to edit the Helm configuration:
   - Using **Form view**
     a. Expand **Root Schema → global → Dynamic plugins configuration → List of dynamic plugins that should be installed in the backstage application**.
     b. Click the **Add List of dynamic plugins that should be installed in the backstage application** link.
     c. In the **Package specification of the dynamic plugin to install. It should be usable by the npm pack command.** field, add the following value: **./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-dynamic**



d. Click **Upgrade**.

- Using **YAML view**

a. Set the value of the **global.dynamic.plugins.package** parameter to **./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-dynamic** as shown in the following example:

```
global:
  dynamic:
    # other Red Hat Developer Hub Helm Chart configurations
    plugins:
      - disabled: false
        package: >-
          ./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-
dynamic
    # other Red Hat Developer Hub Helm Chart configurations
```

b. Click **Upgrade**.

4. Add the following code in the ConfigMap:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app-config-rhdh
data:
  app-config-rhdh.yaml: |
    ...
      catalog:
        providers:
          github:
```

```
        providerId:
          organization: '${GITHUB_ORG}'
          schedule:
            frequency:
              minutes: 30
            initialDelay:
              seconds: 15
            timeout:
              minutes: 3
...
```

In the previous code, replace **${GITHUB_ORG}** with the GitHub organization from where you want to discover the components. Also, if there is a single provider, then following code can be added in the ConfigMap:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app-config-rhdh
data:
  app-config-rhdh.yaml: |
    ...
    catalog:
      providers:
        github:
          organization: ${GITHUB_ORG}
          schedule:
            frequency: { minutes: 1 }
            timeout: { minutes: 1 }
            initialDelay: { seconds: 100 }
    ...
```

The **providerId** in the previous code is required to identify the provider when there is a list of them.

5. Click **Save**.

3.2.4. Enabling GitHub organization member discovery in Red Hat Developer Hub

You can also enable GitHub discoverability for the members of your GitHub organization.

Prerequisites

- You have set up the GitHub integration. For more information, see Section 3.2.2, “Configuring GitHub integration”.

Procedure

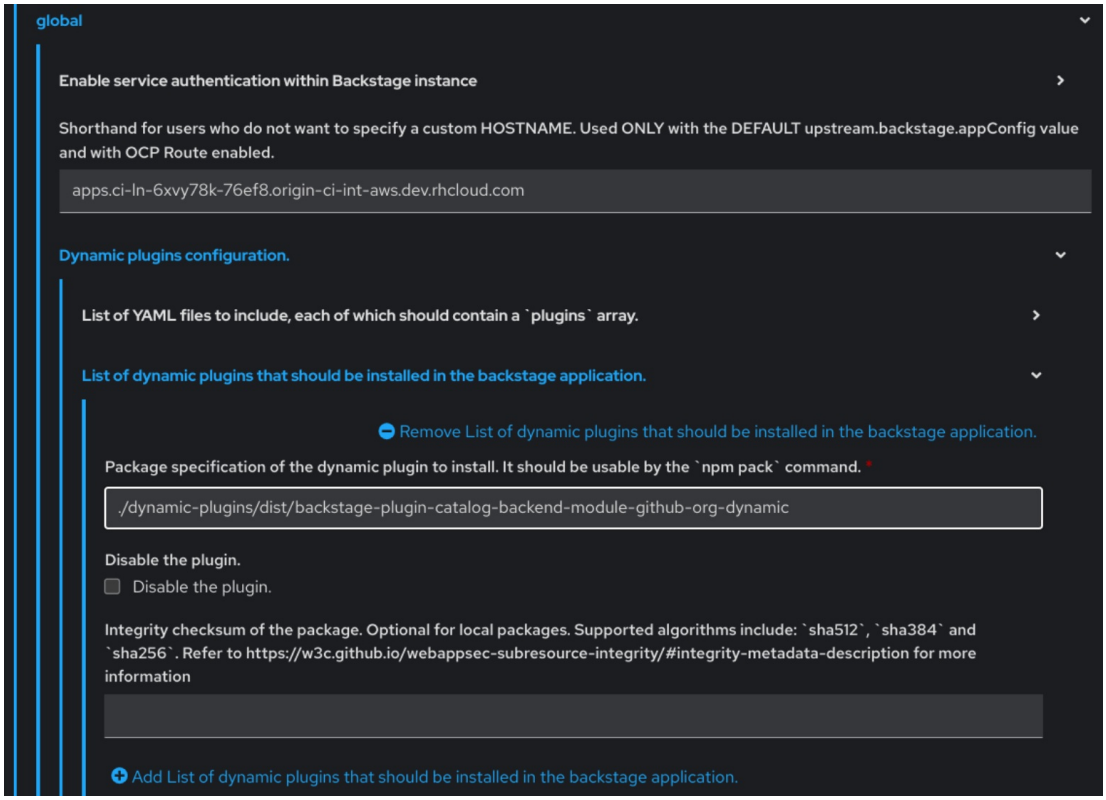
1. In the **Developer** perspective of the OpenShift Container Platform web console, go to the **Helm** tab.
2. Click the overflow menu on a Helm release and select **Upgrade**.
3. Use one of the following views to edit the Helm configuration:

- Using **Form view**

  a. Expand **Root Schema → global → Dynamic plugins configuration → List of dynamic plugins that should be installed in the backstage application**.

  b. Click the **Add List of dynamic plugins that should be installed in the backstage application** link.

  c. In the **Package specification of the dynamic plugin to install. It should be usable by the npm pack command.** field, add the following value: **./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-org-dynamic**



  d. Click **Upgrade**.

- Using **YAML view**

  a. Set the value of the **global.dynamic.plugins.package** parameter to **./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-org-dynamic** as shown in the following example:

```
global:
  dynamic:
    # other Red Hat Developer Hub Helm Chart configurations
    plugins:
      - disabled: false
        package: >-
          ./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-org-
dynamic
    # other Red Hat Developer Hub Helm Chart configurations
```

  b. Click **Upgrade**.

4. Add the following code in the ConfigMap:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app-config-rhdh
data:
  app-config-rhdh.yaml: |
    ...
    catalog:
      providers:
        githubOrg:
          id: production
          githubUrl: "${GITHUB_URL}"
          orgs: [ "${GITHUB_ORG}" ]
    ...
```

where:

${GITHUB_URL}

Denotes a variable that you must replace with the GitHub URL.

${GITHUB_ORG}

Denotes a variable that you must replace with the GitHub organization you want to ingest users from.

5. Click **Save**.

CHAPTER 4. CUSTOMIZING THE HOME PAGE IN RED HAT DEVELOPER HUB

In Red Hat Developer Hub, the Home page data is configurable, which can be passed into the **app-config.yaml** file as a proxy. You can provide the Home page data using the following ways:

- Using JSON files that are hosted or GitHub or GitLab. To access the data from the JSON files, you can add the following code in the **app-config.yaml** file:

```
proxy:
  endpoints:
    # Other Proxies
    # customize developer hub instance
    '/developer-hub':
      target: <DOMAIN_URL> # i.e https://raw.githubusercontent.com/
      pathRewrite:
        '^/api/proxy/developer-hub': <path to json file> # i.e /janus-idp/backstage-showcase/main/packages/app/public/homepage/data.json
      changeOrigin: true
      secure: true
      # Change to "false" in case of using self hosted cluster with a self-signed certificate
      headers:
 <HEADER_KEY>: <HEADER_VALUE> # optional and can be passed as needed i.e Authorization can be passed for private GitHub repo and PRIVATE-TOKEN can be passed for private GitLab repo
```

- Using a separate service that provides the Home page data in JSON format using an API.

NOTE

It is not necessary that the same service provides the Home page and Tech Radar data.

You can use the **red-hat-developer-hub-customization-provider** as an example service, which provides data for both Home page and Tech Radar. The **red-hat-developer-hub-customization-provider** service provides the same data as default Developer Hub data. You can fork the **red-hat-developer-hub-customization-provider** service repository from GitHub and modify it with your own data, if required.

This section describes how you can deploy the **red-hat-developer-hub-customization-provider** service onto the cluster where the Developer Hub Helm Chart is deployed.

Prerequisites

- You have installed the Red Hat Developer Hub using Helm Chart. For more information, see Chapter 2, *Installing Red Hat Developer Hub* .

Procedure

1. In Red Hat OpenShift, select **+Add** and click **Import from Git** option.
2. Add the URL of your Git repository to the **Git Repo URL** field.
   To use the **red-hat-developer-hub-customization-provider** service, you can add the URL of red-hat-developer-hub-customization-provider repository.

3. In the **General** section, rename the value in the **Name** field to **rhdh-customization-provider** and click **Create**.

4. Go to the **Advanced Options** and copy the value from the **Target Port**.
   The **Target Port** is used to automatically generate a Kubernetes or OpenShift service to communicate with.

5. To view the service, navigate to the **OpenShift Administrator** view and go to the **Networking → Service** section.
   You can also view the **Service Resources** in the Topology view.

   If you follow this procedure with examples, then **rhdh-customization-provider** service is called and contains the 8080 port. The provided API URL for the Home page must return the data in JSON format as shown in the following example:

```
[
  {
    "title": "Dropdown 1",
    "isExpanded": false,
    "links": [
      {
        "iconUrl": "https://imagehost.com/image.png",
        "label": "Dropdown 1 Item 1",
        "url": "https://example.com/"
      },
      {
        "iconUrl": "https://imagehost2.org/icon.png",
        "label": "Dropdown 1 Item 2",
        "url": ""
      }
    ]
  },
  {
    "title": "Dropdown 2",
    "isExpanded": true,
    "links": [
      {
        "iconUrl": "http://imagehost3.edu/img.jpg",
        "label": "Dropdown 2 Item 1",
        "url": "http://example.com"
      }
    ]
  }
]
```

   If the request call fails or is not configured, the Developer Hub instance falls back to the default local data.

   To access the Home page in Red Hat Developer Hub, the base URL must include the **/developer-hub** proxy.

6. Add the following code to the **app-config-rhdh.yaml** file:

```
proxy:
  endpoints:
    # Other Proxies
```

```
    # customize developer hub instance
    '/developer-hub':
      target: ${HOMEPAGE_DATA_URL}
      changeOrigin: true
      # Change to "false" in case of using self-hosted cluster with a self-signed certificate
      secure: true
```

Ensure that the API request call returns the response in JSON format.

7. Define the **HOMEPAGE_DATA_URL** as **http://<SERVICE_NAME>:8080. For example, `http://rhdh-customization-provider:8080**.
   You can replace the **HOMEPAGE_DATA_URL** by adding the URL to **rhdh-secrets** or directly replacing it in your custom ConfigMap.

8. Delete the Developer Hub Pod to pull in the changes.
   If the images or icons do not load, then whitelist them by adding your image or icon host URLs to the content security policy's (csp) **img-src** in your custom ConfigMap as follows:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app-config-rhdh
data:
  app-config-rhdh.yaml: |
    app:
      title: Red Hat Developer Hub
    backend:
      csp:
        connect-src:
          - "'self'"
          - 'http:'
          - 'https:'
        img-src:
          - "'self'"
          - 'data:'
          - <image host url 1>
          - <image host url 2>
          - <image host url 3>
    # Other Configurations
```

After that, delete the pod to ensure that the new configurations are loaded correctly.

CHAPTER 5. CUSTOMIZING THE TECH RADAR PAGE IN THE RED HAT DEVELOPER HUB

In Red Hat Developer Hub, the Tech Radar page is not enabled using the dynamic plugin feature in the Helm Chart.

Similar to Home page customization, the base Tech Radar URL must include the **/developer-hub/tech-radar** proxy. You can provide the Tech Radar page data using the following ways:

- Using JSON files that are hosted or GitHub or GitLab. To access the data from the JSON files, you can add the following code in the **app-config.yaml** file:

```
proxy:
  endpoints:
    # Other Proxies
    # customize developer hub instance
    '/developer-hub':
      target: <DOMAIN_URL> # i.e https://raw.githubusercontent.com/
      pathRewrite:
        '^/api/proxy/developer-hub/tech-radar': <path to json file> # i.e /janus-idp/backstage-showcase/main/packages/app/public/tech-radar/data-default.json
  '^/api/proxy/developer-hub': <path to json file> # i.e /janus-idp/backstage-showcase/main/packages/app/public/homepage/data.json
      changeOrigin: true
      secure: true

      # Change to "false" in case of using self hosted cluster with a self-signed certificate
      headers:
 <HEADER_KEY>: <HEADER_VALUE> # optional and can be passed as needed i.e Authorization can be passed for private GitHub repo and PRIVATE-TOKEN can be passed for private GitLab repo
```

NOTE

As overlapping exist between the **pathRewrites** that are used for the **tech-radar** and **homepage** quick access proxies, the configuration for the **tech-radar** (**^api/proxy/developer-hub/tech-radar**) must exist before the configuration for the **homepage** (**^/api/proxy/developer-hub**).

For more information about customizing the Home page in Red Hat Developer Hub, see Chapter 4, *Customizing the Home page in Red Hat Developer Hub* .

- Using a separate service that provides the Tech Radar data in JSON format using an API.

Prerequisites

- You have installed the Red Hat Developer Hub using Helm Chart. For more information, see Chapter 2, *Installing Red Hat Developer Hub* .

Procedure

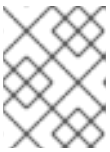
1. Add the following code to the **app-config-rhdh.yaml** file:

```
proxy:
```

```
endpoints:
  # Other Proxies
  '/developer-hub/tech-radar':
    target: ${TECHRADAR_DATA_URL}
    changeOrigin: true
    # Change to "false" in case of using self hosted cluster with a self-signed certificate
    secure: true
```

Ensure that the API request call returns the response in JSON format.

2. Define the **TECHRADAR_DATA_URL`** as **http://<SERVICE_NAME>/tech-radar**, for example **http://rhdh-customization-provider/tech-radar**.



NOTE

You can define the **TECHRADAR_DATA_URL** either by adding it to **rhdh-secrets** or directly replacing it with its value in your custom ConfigMap.

3. Delete the Developer Hub Pod to pull in the changes.

CHAPTER 6. ADDITIONAL CUSTOMIZATIONS IN RED HAT DEVELOPER HUB

This section describes additional customization options that you can apply to the Red Hat Developer Hub.

Customizing tab tooltip

To customize the tab tooltip, add the following content to your **app-config-rhdh.yaml** file:

```
app:
  title: My custom developer hub
```

Customizing branding of your Developer Hub instance

To customize the branding of your Developer Hub instance, add the following content to your **app-config-rhdh.yaml** file:

```
app:
  branding:
    fullLogo: ${BASE64_EMBEDDED_FULL_LOGO}
    iconLogo: ${BASE64_EMBEDDED_ICON_LOGO}
    theme:
      light:
        primaryColor: ${PRIMARY_LIGHT_COLOR}
        headerColor1: ${HEADER_LIGHT_COLOR_1}
        headerColor2: ${HEADER_LIGHT_COLOR_2}
        navigationIndicatorColor: ${NAV_INDICATOR_LIGHT_COLOR}
      dark:
        primaryColor: ${PRIMARY_DARK_COLOR}
        headerColor1: ${HEADER_DARK_COLOR_1}
        headerColor2: ${HEADER_DARK_COLOR_2}
        navigationIndicatorColor: ${NAV_INDICATOR_DARK_COLOR}
```

In the previous configuration,

- **fullLogo** is the logo on the expanded (pinned) sidebar and expects a base64 encoded image.
- **iconLogo** is the logo on the collapsed (unpinned) sidebar and expects a base64 encoded image.
- **primaryColor** is the color of links and most buttons to the inputted color. The supported formats for **primaryColor** include:
  - **#nnn**
  - **#nnnnnn**
  - **rgb()**
  - **rgba()**
  - **hsl()**
  - **hsla()**

    - **color()**

- **headerColor1** (left-side of the banner) and **headerColor2** (right-side of the banner) changes the color of the header banner of each page, as well as the banner for template cards. The supported formats for **headerColor1** and **headerColor2** include:
    - **#nnn**
    - **#nnnnnn**
    - **rgb()**
    - **rgba()**
    - **hsl()**
    - **hsla()**
    - **color()**
- **navigationIndicatorColor** changes the color of the indicator in the sidebar that indicates which tab you are on. The supported formats for **navigationIndicatorColor** include:
    - **#nnn**
    - **#nnnnnn**
    - **rgb()**
    - **rgba()**
    - **hsl()**
    - **hsla()**
    - **color()**

CHAPTER 7. CUSTOMIZING YOUR THEME IN RED HAT DEVELOPER HUB

You can customize your Red Hat Developer Hub (Developer Hub) theme mode.

RHDH supports the following theme modes:

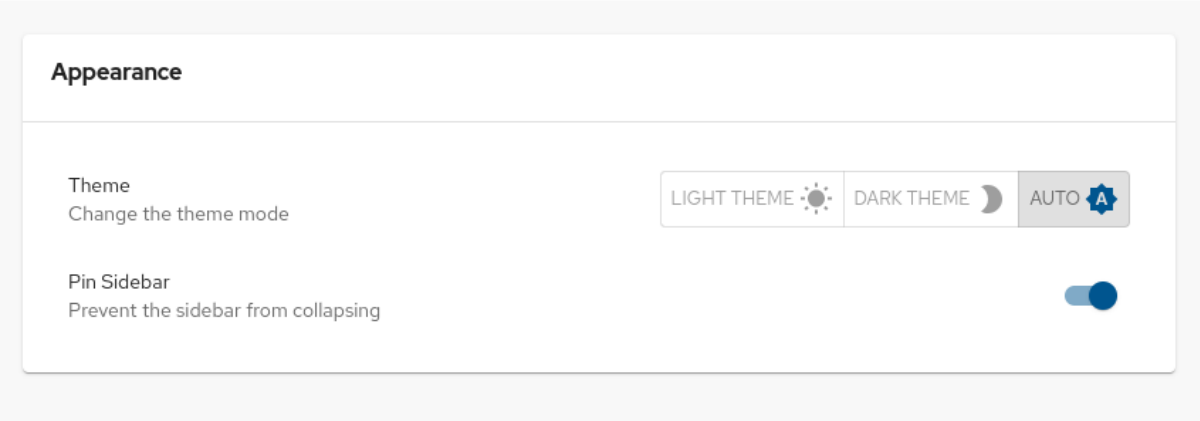
- Light theme (default)
- Dark theme
- Auto

Prerequisites

- You are logged in to the RHDH web console.

Procedure

1. Click **Settings**.
2. From the **Appearance** panel, click **LIGHT THEME**, **DARK THEME**, or **AUTO** to change the theme mode.



CHAPTER 8. SERVICENOW CUSTOM ACTIONS 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.

In Red Hat Developer Hub, you can access ServiceNow custom actions (custom actions) for fetching and registering resources in the catalog.

The custom actions in Developer Hub enable you to facilitate and automate the management of records. Using the custom actions, you can perform the following actions:

- Create, update, or delete a record
- Retrieve information about a single record or multiple records

8.1. ENABLING SERVICENOW CUSTOM ACTIONS PLUGIN IN RED HAT DEVELOPER HUB

In Red Hat Developer Hub, the ServiceNow custom actions are provided as a pre-loaded plugin, which is disabled by default. You can enable the custom actions plugin using the following procedure.

Prerequisites

- Red Hat Developer Hub is installed and running. For more information about installing the Developer Hub, see Chapter 2, *Installing Red Hat Developer Hub* .
- You have created a project in the Developer Hub.

Procedure

1. To activate the custom actions plugin, add a **package** with plugin name and update the **disabled** field in your Helm Chart as follows:

```
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: ./dynamic-plugins/dist/janus-idp-backstage-scaffolder-backend-module-
servicenow-dynamic
        disabled: false
```

NOTE

The default configuration for a plugin is extracted from the **dynamic-plugins.default.yaml** file, however, you can use a **pluginConfig** entry to override the default configuration.

2. Set the following variables in the Helm Chart to access the custom actions:

```
servicenow:
  # The base url of the ServiceNow instance.
  baseUrl: ${SERVICENOW_BASE_URL}
  # The username to use for authentication.
  username: ${SERVICENOW_USERNAME}
  # The password to use for authentication.
  password: ${SERVICENOW_PASSWORD}
```

8.2. SUPPORTED SERVICENOW CUSTOM ACTIONS IN RED HAT DEVELOPER HUB

The ServiceNow custom actions enable you to manage records in the Red Hat Developer Hub. The custom actions support the following HTTP methods for API requests:

- **GET**: Retrieves specified information from a specified resource endpoint
- **POST**: Creates or updates a resource
- **PUT**: Modify a resource
- **PATCH**: Updates a resource
- **DELETE**: Deletes a resource

8.2.1. ServiceNow custom actions

[GET] servicenow:now:table:retrieveRecord

Retrieves information of a specified record from a table in the Developer Hub.

Table 8.1. Input parameters

Name	Type	Requirement	Description
tableName	**string**	Required	Name of the table to retrieve the record from
sysId	**string**	Required	Unique identifier of the record to retrieve
sysparmDisplayValue	**enum("true", "false", "all")**	Optional	Returns field display values such as **true**, actual values as **false**, or both. The default value is **false**.

Name	Type	Requirement	Description
sysparmExcludeReferenceLink	**boolean**	Optional	Set as **true** to exclude Table API links for reference fields. The default value is **false**.
sysparmFields	**string[]**	Optional	Array of fields to return in the response
sysparmView	**string**	Optional	Renders the response according to the specified UI view. You can override this parameter using **sysparm_fields**.
sysparmQueryNoDomain	**boolean**	Optional	Set as **true** to access data across domains if authorized. The default value is **false**.

Table 8.2. Output parameters

Name	Type	Description
result	**Record<PropertyKey, unknown>**	The response body of the request

[GET] servicenow:now:table:retrieveRecords

Retrieves information about multiple records from a table in the Developer Hub.

Table 8.3. Input parameters

Name	Type	Requirement	Description
tableName	**string**	Required	Name of the table to retrieve the records from
sysparamQuery	**string**	Optional	Encoded query string used to filter the results
sysparmDisplayValue	**enum("true", "false", "all")**	Optional	Returns field display values such as **true**, actual values as **false**, or both. The default value is **false**.
sysparmExcludeReferenceLink	**boolean**	Optional	Set as **true** to exclude Table API links for reference fields. The default value is **false**.

Name	Type	Requirement	Description
sysparmSuppressPaginationHeader	**boolean**	Optional	Set as **true** to suppress pagination header. The default value is **false**.
sysparmFields	**string[]**	Optional	Array of fields to return in the response
sysparmLimit	**int**	Optional	Maximum number of results returned per page. The default value is **10,000**.
sysparmView	**string**	Optional	Renders the response according to the specified UI view. You can override this parameter using **sysparm_fields**.
sysparmQueryCategory	**string**	Optional	Name of the query category to use for queries
sysparmQueryNoDomain	**boolean**	Optional	Set as **true** to access data across domains if authorized. The default value is **false**.
sysparmNoCount	**boolean**	Optional	Does not execute a select count(*) on the table. The default value is **false**.

Table 8.4. Output parameters

Name	Type	Description
result	**Record<PropertyKey, unknown>**	The response body of the request

[POST] servicenow:now:table:createRecord

Creates a record in a table in the Developer Hub.

Table 8.5. Input parameters

Name	Type	Requirement	Description
tableName	**string**	Required	Name of the table to save the record in

Name	Type	Requirement	Description
requestBody	**Record<PropertyKey, unknown>**	Optional	Field name and associated value for each parameter to define in the specified record
sysparmDisplayValue	**enum("true", "false", "all")**	Optional	Returns field display values such as **true**, actual values as **false**, or both. The default value is **false**.
sysparmExcludeReferenceLink	**boolean**	Optional	Set as **true** to exclude Table API links for reference fields. The default value is **false**.
sysparmFields	**string[]**	Optional	Array of fields to return in the response
sysparmInputDisplayValue	**boolean**	Optional	Set field values using their display value such as **true** or actual value as **false**. The default value is **false**.
sysparmSuppressAutoSysField	**boolean**	Optional	Set as **true** to suppress auto-generation of system fields. The default value is **false**.
sysparmView	**string**	Optional	Renders the response according to the specified UI view. You can override this parameter using **sysparm_fields**.

Table 8.6. Output parameters

Name	Type	Description
result	**Record<PropertyKey, unknown>**	The response body of the request

[PUT] servicenow:now:table:modifyRecord

Modifies a record in a table in the Developer Hub.

Table 8.7. Input parameters

Name	Type	Requirement	Description
tableName	**string**	Required	Name of the table to modify the record from
sysId	**string**	Required	Unique identifier of the record to modify

Name	Type	Requirement	Description
requestBody	**Record<PropertyKey, unknown>**	Optional	Field name and associated value for each parameter to define in the specified record
sysparmDisplayValue	**enum("true", "false", "all")**	Optional	Returns field display values such as **true**, actual values as **false**, or both. The default value is **false**.
sysparmExcludeReferenceLink	**boolean**	Optional	Set as **true** to exclude Table API links for reference fields. The default value is **false**.
sysparmFields	**string[]**	Optional	Array of fields to return in the response
sysparmInputDisplayValue	**boolean**	Optional	Set field values using their display value such as **true** or actual value as **false**. The default value is **false**.
sysparmSuppressAutoSysField	**boolean**	Optional	Set as **true** to suppress auto-generation of system fields. The default value is **false**.
sysparmView	**string**	Optional	Renders the response according to the specified UI view. You can override this parameter using **sysparm_fields**.
sysparmQueryNoDomain	**boolean**	Optional	Set as **true** to access data across domains if authorized. The default value is **false**.

Table 8.8. Output parameters

Name	Type	Description
result	**Record<PropertyKey, unknown>**	The response body of the request

[PATCH] servicenow:now:table:updateRecord

Updates a record in a table in the Developer Hub.

Table 8.9. Input parameters

Name	Type	Requirement	Description
tableName	**string**	Required	Name of the table to update the record in
sysId	**string**	Required	Unique identifier of the record to update
requestBody	**Record<PropertyKey, unknown>**	Optional	Field name and associated value for each parameter to define in the specified record
sysparmDisplayValue	**enum("true", "false", "all")**	Optional	Returns field display values such as **true**, actual values as **false**, or both. The default value is **false**.
sysparmExcludeReferenceLink	**boolean**	Optional	Set as **true** to exclude Table API links for reference fields. The default value is **false**.
sysparmFields	**string[]**	Optional	Array of fields to return in the response
sysparmInputDisplayValue	**boolean**	Optional	Set field values using their display value such as **true** or actual value as **false**. The default value is **false**.
sysparmSuppressAutoSysField	**boolean**	Optional	Set as **true** to suppress auto-generation of system fields. The default value is **false**.
sysparmView	**string**	Optional	Renders the response according to the specified UI view. You can override this parameter using **sysparm_fields**.
sysparmQueryNoDomain	**boolean**	Optional	Set as **true** to access data across domains if authorized. The default value is **false**.

Table 8.10. Output parameters

Name	Type	Description
result	**Record<PropertyKey, unknown>**	The response body of the request

[DELETE] servicenow:now:table:deleteRecord

Deletes a record from a table in the Developer Hub.

Table 8.11. Input parameters

Name	Type	Requirement	Description
tableName	**string**	Required	Name of the table to delete the record from
sysId	**string**	Required	Unique identifier of the record to delete
sysparmQueryNoDomain	**boolean**	Optional	Set as **true** to access data across domains if authorized. The default value is **false**.

CHAPTER 9. GITHUB AUTHENTICATION PROVIDER

Red Hat Developer Hub uses a built-in GitHub authentication provider to authenticate users in GitHub or GitHub Enterprise.

9.1. GITHUB APP OVERVIEW

GitHub Apps are generally preferred to OAuth apps because they use fine-grained permissions, give more control over which repositories the application can access, and use short-lived tokens. For more information, see GitHub Apps overview in the GitHub documentation.

9.2. REGISTERING A GITHUB APP

In a GitHub App, you configure the allowed scopes as part of that application, therefore, you must verify the scope that your plugins require. The scope information is available in the plugin README files.

To add GitHub authentication, complete the steps in Registering a GitHub App on the GitHub website.

Use the following examples to enter the information about your production environment into the required fields on the **Register new GitHub App** page:

- Application name: Red Hat Developer Hub
- Homepage URL: **https://developer-hub-<NAMESPACE_NAME>.<KUBERNETES_ROUTE_HOST>**
- Authorization callback URL: **https://developer-hub-<NAMESPACE_NAME>.<KUBERNETES_ROUTE_HOST>/api/auth/github/handler/frame**

NOTE

The Homepage URL points to the Developer Hub front end, while the authorization callback URL points to the authentication provider backend.

9.3. CONFIGURING A GITHUB APP IN DEVELOPER HUB

To add GitHub authentication for Developer Hub, you must configure the GitHub App in your **app-config.yaml** file.

The GitHub authentication provider uses the following configuration keys:

- **clientId**: the client ID that you generated on GitHub. For example: b59241722e3c3b4816e2
- **clientSecret**: the client secret tied to the generated client ID.
- **enterpriseInstanceUrl** (optional): the base URL for a GitHub Enterprise instance. For example: **https://ghe.<company>.com**. The **enterpriseInstanceUrl** is only needed for GitHub Enterprise.
- **callbackUrl** (optional): the callback URL that GitHub uses when initiating an OAuth flow. For example: https://your-intermediate-service.com/handler. The **callbackUrl** is only needed if Developer Hub is not the immediate receiver, such as in cases when you use one OAuth app for many Developer Hub instances.

To configure the GitHub App, add the provider configuration to your **app-config.yaml** file under the root auth configuration. For example:

```
auth:
 environment: production
 providers:
   github:
     production:
       clientId: ${GITHUB_APP_CLIENT_ID}
       clientSecret: ${GITHUB_APP_CLIENT_SECRET}
       ## uncomment if using GitHub Enterprise
       # enterpriseInstanceUrl: ${GITHUB_URL}
```

9.4. ADDING THE GITHUB PROVIDER TO THE DEVELOPER HUB FRONT END

To add the provider to the front end, add the sign in configuration to your **app-config.yaml** file. For example:

```
signInPage: github
```

Additional resources

- For information about authenticating Backstage access with GitHub, see GitHub Authentication Provider in the community documentation.
- For information about adding the provider to the Backstage front end, see Enabling authentication in Showcase in the community documentation.

CHAPTER 10. OPENID CONNECT AUTHENTICATION PROVIDER

Red Hat Developer Hub uses the OpenID Connect (OIDC) authentication provider to authenticate with third-party services that support the OIDC protocol.

10.1. OVERVIEW OF USING THE OIDC AUTHENTICATION PROVIDER IN DEVELOPER HUB

You can configure the OIDC authentication provider in Developer Hub by updating your **app-config.yaml** file under the root auth configuration. For example:

```
auth:
  environment: production
  # Providing an auth.session.secret will enable session support in the auth-backend
  session:
    secret: ${SESSION_SECRET}
  providers:
    oidc:
      production:
        metadataUrl: ${AUTH_OIDC_METADATA_URL}
        clientId: ${AUTH_OIDC_CLIENT_ID}
        clientSecret: ${AUTH_OIDC_CLIENT_SECRET}
        prompt: ${AUTH_OIDC_PROMPT} # Recommended to use auto
        ## Uncomment for additional configuration options
        # callbackUrl: ${AUTH_OIDC_CALLBACK_URL}
        # tokenEndpointAuthMethod: ${AUTH_OIDC_TOKEN_ENDPOINT_METHOD}
        # tokenSignedResponseAlg: ${AUTH_OIDC_SIGNED_RESPONSE_ALG}
        # scope: ${AUTH_OIDC_SCOPE}

  signInPage: oidc
```

10.2. CONFIGURING KEYCLOAK WITH THE OIDC AUTHENTICATION PROVIDER

Red Hat Developer Hub includes an OIDC authentication provider that can authenticate users by using Keycloak.

IMPORTANT

The user that you create in Keycloak must also be available in the Developer Hub catalog.

Procedure

1. In Keycloak, create a new realm, for example **RHDH**.
2. Add a new user.

   Username

   Username for the user, for example: **rhdhuser**

   Email

Email address of the user.

First name

First name of the user.

Last name

Last name of the user.

Email verified

Toggle to **On**.

3. Click **Create**.
4. Navigate to the **Credentials** tab.
5. Click **Set password**.
6. Enter the **Password** for the user account and toggle **Temporary** to **Off**.
7. Create a new Client ID, for example, **RHDH**.

Client authentication

Toggle to **On**.

Valid redirect URIs

Set to the OIDC handler URL, for example, **https://<RHDH_URL>/api/auth/oidc/handler/frame**.

8. Navigate to the **Credentials** tab and copy the **Client secret**.
9. Save the Client ID and the Client Secret for the next step.
10. In Developer Hub, add your Keycloak credentials in your Developer Hub secrets.
    a. Edit your Developer Hub secrets, such as secrets-rhdh.
    b. Add the following key/value pairs:

AUTH_KEYCLOAK_CLIENT_ID

Enter the Client ID that you generated in Keycloak, such as **RHDH**.

AUTH_KEYCLOAK_CLIENT_SECRET

Enter the Client Secret that you generated in Keycloak.

11. Set up the OIDC authentication provider in your Developer Hub custom configuration.
    a. Edit your custom Developer Hub ConfigMap, such as **app-config-rhdh**.
    b. In the **app-config-rhdh.yaml** content, add the **oidc** provider configuration under the root **auth** configuration, and enable the **oidc** provider for sign-in:

app-config-rhdh.yaml fragment

```
auth:
  environment: production
  providers:
    oidc:
      production:
```

```
      clientId: ${AUTH_KEYCLOAK_CLIENT_ID}
      clientSecret: ${AUTH_KEYCLOAK_CLIENT_SECRET}
      metadataUrl: ${KEYCLOAK_BASE_URL}/auth/realms/${KEYCLOAK_REALM}
      prompt: ${KEYCLOAK_PROMPT} # recommended to use auto
      ## Uncomment for additional configuration options
      #callbackUrl: ${KEYCLOAK_CALLBACK_URL}
      #tokenEndpointAuthMethod: ${KEYCLOAK_TOKEN_ENDPOINT_METHOD}
      #tokenSignedResponseAlg: ${KEYCLOAK_SIGNED_RESPONSE_ALG}
      #scope: ${KEYCLOAK_SCOPE}

  signInPage: oidc
```

Verification

1. Restart your **backstage-developer-hub** application to apply the changes.
2. Your Developer Hub sign-in page displays **Sign in using OIDC**.

10.3. MIGRATING FROM OAUTH2 PROXY WITH KEYCLOAK TO OIDC IN DEVELOPER HUB

If you are using OAuth2 Proxy as an authentication provider with Keycloak, and you want to migrate to OIDC, you can update your authentication provider configuration to use OIDC.

Procedure

1. In Keycloak, update the valid redirect URI to **https://<rhdh_url>/api/auth/oidc/handler/frame**. Make sure to replace **<rhdh_url>** with your Developer Hub application URL, such as, **my.rhdh.example.com**.
2. Replace the **oauth2Proxy** configuration values in the **auth** section of your **app-config.yaml** file with the **oidc** configuration values.
3. Update the **signInPage** configuration value from **oauth2Proxy** to **oidc**.
   The following example shows the **auth.providers** and **signInPage** configuration for **oauth2Proxy** prior to migrating the authentication provider to **oidc**:

```
auth:
  environment: production
  session:
    secret: ${SESSION_SECRET}
  providers:
    oauth2Proxy: {}

signInPage: oauth2Proxy
```

   The following example shows the **auth.providers** and **signInPage** configuration after migrating the authentication provider to **oidc**:

```
auth:
  environment: production
  session:
    secret: ${SESSION_SECRET}
  providers:
```

```
    oidc:
      production:
        metadataUrl: ${KEYCLOAK_METADATA_URL}
        clientId: ${KEYCLOAK_CLIENT_ID}
        clientSecret: ${KEYCLOAK_CLIENT_SECRET}
        prompt: ${KEYCLOAK_PROMPT} # recommended to use auto

  signInPage: oidc
```

4. Remove the OAuth2 Proxy sidecar container and update the **upstream.service** section of your Helm chart's **values.yaml** file as follows:

   - **service.ports.backend**: **7007**
   - **service.ports.targetPort**: **backend**
     The following example shows the **service** configuration for **oauth2Proxy** prior to migrating the authentication provider to **oidc**:

     ```
     service:
       ports:
         name: http-backend
         backend: 4180
         targetPort: oauth2Proxy
     ```

     The following example shows the **service** configuration after migrating the authentication provider to **oidc**:

     ```
     service:
       ports:
         name: http-backend
         backend: 7007
         targetPort: backend
     ```

5. Upgrade the Developer Hub Helm chart.