Red Hat Developer Hub 1.5

Customizing Red Hat Developer Hub

Customize Red Hat Developer Hub appearance and features, such as templates, Learning Paths, Tech Radar, Home page, and quick access cards.

Last Updated: 2025-07-15

Red Hat Developer Hub 1.5 Customizing Red Hat Developer Hub

Customize Red Hat Developer Hub appearance and features, such as templates, Learning Paths, Tech Radar, Home page, and quick access cards.

Legal Notice

Abstract

Authorized users can customize Red Hat Developer Hub appearance and features, such as templates, Learning Paths, Tech Radar, Home page, and quick access cards.

Table of Contents

PREFACE

Authorized users can customize Red Hat Developer Hub appearance and features, such as templates, Learning Paths, Tech Radar, Home page, and quick access cards.

CHAPTER 1. CUSTOMIZING YOUR RED HAT DEVELOPER HUB TITLE

You can change the default Red Hat Developer Hub display name.

Prerequisites

- Custom Developer Hub configuration .

Procedure

- In your custom **app-config.yaml** file, enter your Developer Hub instance display name, such as *<Red Hat Developer Hub>*.

  app-config.yaml excerpt

  ```
  app:
    title: My custom Red Hat Developer Hub title
  ```

CHAPTER 2. CUSTOMIZING YOUR RED HAT DEVELOPER HUB BASE URL

You can change the default Red Hat Developer Hub base URL.

Prerequisites

- You know your desired Developer Hub external URL: https://*<my_developer_hub_url>*, and have configured DNS to point to your Red Hat OpenShift Container Platform cluster.
- Custom Developer Hub configuration .

Procedure

- In your custom **app-config.yaml** file, enter your Developer Hub external URL, such as https://*<my_developer_hub_url>*.

  app-config.yaml excerpt

  ```
  app:
    baseUrl: https://<my_developer_hub_url>
  backend:
    baseUrl: https://<my_developer_hub_url>
    cors:
      origin: https://<my_developer_hub_url>
  ```

CHAPTER 3. CUSTOMIZING RED HAT DEVELOPER HUB BACKEND SECRET

The default Red Hat Developer Hub configuration defines the Developer Hub backend secret for service to service authentication.

You can define your custom Developer Hub backend secret.

Prerequisites

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

Procedure

1. To define the Developer Hub backend secret, add to your custom ***<my_product_secrets>*.txt** file the **BACKEND_SECRET** environment variable with a base64 encoded string. Use a unique value for each Developer Hub instance.

```
$ echo > <my_product_secrets>.txt "BACKEND_SECRET=$(node -p
'require("crypto").randomBytes(24).toString("base64")')"
```

<my_product_secrets>.txt example

```
BACKEND_SECRET=3E2/rIPuZNFCtYHoxVP8wjriffnN1q/z
```

2. Add your backend secret to your custom **app-config.yaml** file.

app-config.yaml excerpt defining the backend secret

```
backend:
  auth:
    externalAccess:
      - type: legacy
        options:
          subject: legacy-default-config
          secret: "${BACKEND_SECRET}"
```

CHAPTER 4. CONFIGURING TEMPLATES

Configure templates to create software components, and publish these components to different locations, such as the Red Hat Developer Hub software catalog, or Git repositories.

A template is a form composed of different UI fields that is defined in a YAML file. Templates include *actions*, which are steps that are executed in sequential order and can be executed conditionally.

4.1. CREATING A SOFTWARE TEMPLATE BY USING THE TEMPLATE EDITOR

Use the Red Hat Developer Hub Template Editor to create a Software Template.

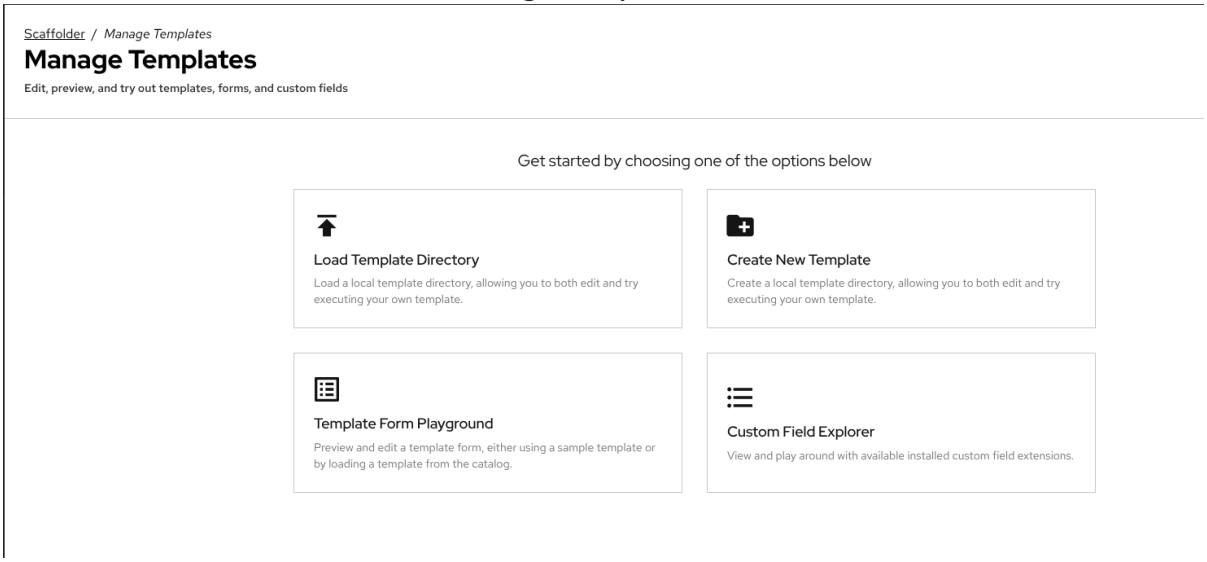
Alternately, you can use the Template Editor to do any of the following actions:

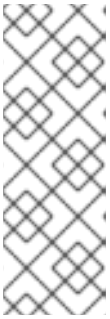
- **File > Open template directory**
- **File > Create template directory**
- **File > Close template editor**
- Use the **Custom Fields Explorer** button to test custom fields in your **templates.yaml** file
- View **Installed Actions Documentation**

Procedure

To create a Software Template by using the Template Editor templates, complete the following steps:

1. In your Red Hat Developer Hub navigation menu, click **Create...**.
2. Click the overview menu and select **Manage Templates**.





NOTE

- The following options on the **Managed Templates** page do not create a software component in your Red Hat Developer Hub instance:
  - **Template Form Playground** Use to create and test the **templates.yaml** file
  - **Custom Field Explorer**: Use to test custom fields

3. On the **Managed Templates** page, select any of the following options:
   - **Load Template Directory**: Use to load an existing **templates.yaml** file
     - In your local file manager, navigate to the folder where your **templates.yaml** file is stored and click **Select**.
   - **Create New Template**: Use to create a **templates.yaml** file
     a. In your local file manager, navigate to the folder where you want the Template Editor to create a **templates.yaml** file and click **Select**.
     b. On the **Template Editor** page, select the **templates.yaml** file.
     c. (Optional) You can preview and test your template specifications.
        i. On the **Fill in some steps** tab, enter text into the required fields and click **Next**.
        ii. On the **Repository Location** tab, enter text into the required fields and click **Review**.
        iii. Modify the YAML definition for the parameters of your template. For more information about these parameters, see Section 4.2, "Creating a Software Template as a YAML file".
        iv. Review the information for accuracy, then click **Create**.

Verification

1. Click the **Catalog** tab in the navigation panel.
2. In the **Kind** drop-down menu, select **Template**.
3. Confirm that your Software Template is shown in the list of existing templates.

Next step

- Import your Software Template in your RHDH instance .

4.2. CREATING A SOFTWARE TEMPLATE AS A YAML FILE

You can create a Software Template by defining a **Template** object as a YAML file.

The **Template** object describes the template and its metadata. It also contains required input variables and a list of actions that are executed by the scaffolding service.

Template object example

```
apiVersion: scaffolder.backstage.io/v1beta3
kind: Template
metadata:
  name: template-name (1)
  title: Example template (2)
  description: An example template for v1beta3 scaffolder. (3)
spec:
  owner: backstage/techdocs-core (4)
  type: service (5)
  parameters: (6)
    - title: Fill in some steps
      required:
        - name
      properties:
        name:
          title: Name
          type: string
          description: Unique name of the component
        owner:
          title: Owner
          type: string
          description: Owner of the component
    - title: Choose a location
      required:
        - repoUrl
      properties:
        repoUrl:
          title: Repository Location
          type: string
  steps: (7)
    - id: fetch-base
      name: Fetch Base
      action: fetch:template
      # ...
  output: (8)
    links:
      - title: Repository (9)
        url: ${{ steps['publish'].output.remoteUrl }}
      - title: Open in catalog (10)
        icon: catalog
        entityRef: ${{ steps['register'].output.entityRef }}
# ...
```

(1) Specify a name for the Software Template.

(2) Specify a title for the Software Template. This is the title that is visible on the Software Template tile in the **Create...** view.

(3) Specify a description for the Software Template. This is the description that is visible on the Software Template tile in the **Create...** view.

(4) Specify the ownership of the Software Template. The **owner** field provides information about who is responsible for maintaining or overseeing the Software Template within the system or organization. In the provided example, the **owner** field is set to **backstage/techdocs-core**. This

means that this Software Template belongs to the **techdocs-core** project in the **backstage** namespace.

5 Specify the component type. Any string value is accepted for this required field, but your organization should establish a proper taxonomy for these. Red Hat Developer Hub instances may read this field and behave differently depending on its value. For example, a **website** type component may present tooling in the Red Hat Developer Hub interface that is specific to just websites.

The following values are common for this field:

service
: A backend service, typically exposing an API.

website
: A website.

library
: A software library, such as an npm module or a Java library.

6 Use the **parameters** section to specify parameters for user input that are shown in a form view when a user creates a component by using the Software Template in the Red Hat Developer Hub console. Each **parameters** subsection, defined by a title and properties, creates a new form page with that definition.

7 Use the **steps** section to specify steps that are executed in the backend. These steps must be defined by using a unique step ID, a name, and an action. You can view actions that are available on your Red Hat Developer Hub instance by visiting the URL **https://<rhdh_url>/create/actions**.

8 Use the **output** section to specify the structure of output data that is created when the Software Template is used. The **output** section, particularly the **links** subsection, provides valuable references and URLs that users can utilize to access and interact with components that are created from the Software Template.

9 Provides a reference or URL to the repository associated with the generated component.

10 Provides a reference or URL that allows users to open the generated component in a catalog or directory where various components are listed.

Additional resources

- Backstage documentation - Writing Templates
- Backstage documentation - Builtin actions
- Backstage documentation - Writing Custom Actions

4.3. IMPORTING AN EXISTING SOFTWARE TEMPLATE TO RED HAT DEVELOPER HUB

You can add an existing Software Template to your Red Hat Developer Hub instance by using the Catalog Processor.

Prerequisites

- You have created a directory or repository that contains at least one Software Template YAML file.

- If you want to use a Software Template that is stored in a repository such as GitHub or GitLab, you must configure a Red Hat Developer Hub integration for your provider.

Procedure

- In the **app-config.yaml** configuration file, modify the **catalog.rules** section to include a rule for templates, and configure the **catalog.locations** section to point to the Software Template that you want to add, as shown in the following example:

```
# ...
catalog:
  rules:
    - allow: [Template] (1)
  locations:
    - type: url (2)
      target: https://<repository_url>/example-template.yaml (3)
# ...
```

(1) To allow new Software Templates to be added to the catalog, you must add a **Template** rule.

(2) If you are importing templates from a repository, such as GitHub or GitLab, use the **url** type.

(3) Specify the URL for the Software Template.

Verification

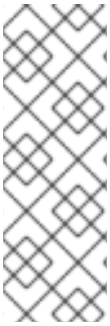
1. Click the **Catalog** tab in the navigation panel.
2. In the **Kind** drop-down menu, select **Template**.
3. Confirm that your Software Template is shown in the list of existing Software Templates.

Additional resources

- Enabling the GitHub authentication provider

CHAPTER 5. CUSTOMIZING THE LEARNING PATHS IN RED HAT DEVELOPER HUB

In Red Hat Developer Hub, you can configure Learning Paths by passing the data into the **app-config.yaml** file as a proxy. The base URL must include the **/developer-hub/learning-paths** proxy.



NOTE

Due to the use of overlapping **pathRewrites** for both the **learning-path** and **homepage** quick access proxies, you must create the **learning-paths** configuration (**^api/proxy/developer-hub/learning-paths**) before you create the **homepage** configuration (**^/api/proxy/developer-hub**).

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

You can provide data to the Learning Path from the following sources:

- JSON files hosted on GitHub or GitLab.
- A dedicated service that provides the Learning Path data in JSON format using an API.

5.1. USING HOSTED JSON FILES TO PROVIDE DATA TO THE LEARNING PATHS

Prerequisites

You have installed Red Hat Developer Hub by using either the Operator or Helm chart. For more information, see Installing Red Hat Developer Hub on OpenShift Container Platform .

Procedure

To access the data from the JSON files, complete the following step:

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

```
proxy:
  endpoints:
    '/developer-hub':
      target: https://raw.githubusercontent.com/
      pathRewrite:
        '^/api/proxy/developer-hub/learning-paths': '/redhat-developer/rhdh/main/packages/app/public/learning-paths/data.json'
        '^/api/proxy/developer-hub/tech-radar': '/redhat-developer/rhdh/main/packages/app/public/tech-radar/data-default.json'
        '^/api/proxy/developer-hub': '/redhat-developer/rhdh/main/packages/app/public/homepage/data.json'
      changeOrigin: true
      secure: true
```

5.2. USING A DEDICATED SERVICE TO PROVIDE DATA TO THE LEARNING PATHS

When using a dedicated service, you can do the following:

- Use the same service to provide the data to all configurable Developer Hub pages or use a different service for each page.
- Use the **red-hat-developer-hub-customization-provider** as an example service, which provides data for both the Home and Tech Radar pages. 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.
- Deploy the **red-hat-developer-hub-customization-provider** service and the Developer Hub Helm chart on the same cluster.

Prerequisites

- You have installed the Red Hat Developer Hub using Helm chart. For more information, see Installing Red Hat Developer Hub on OpenShift Container Platform .

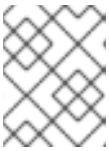
Procedure

To use a dedicated service to provide the Learning Path data, complete the following steps:

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

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

   where the **LEARNING_PATH_DATA_URL** is defined as **http://<SERVICE_NAME>/learning-paths**, for example, **http://rhdh-customization-provider/learning-paths**.

   NOTE

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

2. Delete the Developer Hub pod to ensure that the new configurations are loaded correctly.

CHAPTER 6. CONFIGURING THE GLOBAL HEADER IN RED HAT DEVELOPER HUB

As an administrator, you can configure the Red Hat Developer Hub global header to create a consistent and flexible navigation bar across your Developer Hub instance. By default, the Developer Hub global header includes the following components:

- **Create** button provides quick access to a variety of templates, enabling users to efficiently set up services, backend and front-end plugins within Developer Hub
- **Support** button that can link an internal or external support page
- **Notifications** button displays alerts and updates from plugins and external services
- **Search** input field allows users to find services, components, documentation, and other resources within Developer Hub
- **Plugin extension capabilities**provide a preinstalled and enabled catalog of available plugins in Developer Hub
- **User profile** drop-down menu provides access to profile settings, appearance customization, Developer Hub metadata, and a logout button

6.1. CUSTOMIZING YOUR RED HAT DEVELOPER HUB GLOBAL HEADER

You can use the **red-hat-developer-hub.backstage-plugin-global-header** dynamic plugin to extend the global header with additional buttons and customize the order and position of icons and features. Additionally, you can create and integrate your custom dynamic header plugins using the mount points provided by this new header feature, allowing you to further tailor to suit your needs. For more information on enabling dynamic plugins, see Installing and viewing plugins in Red Hat Developer Hub .

Default global header configuration

```
- package: ./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-global-header
  disabled: false
  pluginConfig:
    app:
      sidebar:
        search: false (1)
        settings: false (2)
    dynamicPlugins:
      frontend:
        default.main-menu-items: (3)
          menuItems:
            default.create:
              title: ''
        red-hat-developer-hub.backstage-plugin-global-header: # the default enabled dynamic header plugin
          mountPoints:
            - mountPoint: application/header
              importName: GlobalHeader
              config:
                position: above-main-content (4)
```

```
    - mountPoint: global.header/component
      importName: SearchComponent
      config:
        priority: 100
    - mountPoint: global.header/component
      importName: Spacer
      config:
        priority: 99
        props:
          growFactor: 0
    - mountPoint: global.header/component
      importName: HeaderIconButton
      config:
        priority: 90
        props:
          title: Create...
          icon: add
          to: create
    - mountPoint: global.header/component
      importName: SupportButton
      config:
        priority: 80
    - mountPoint: global.header/component
      importName: NotificationButton
      config:
        priority: 70
    - mountPoint: global.header/component
      importName: Divider
      config:
        priority: 50
    - mountPoint: global.header/component
      importName: ProfileDropdown
      config:
        priority: 10
    - mountPoint: global.header/profile
      importName: MenuItemLink
      config:
        priority: 100
        props:
          title: Settings
          link: /settings
          icon: manageAccounts
    - mountPoint: global.header/profile
      importName: LogoutButton
      config:
        priority: 10
```

1. **search**: Hides the **Search** modal in the sidebar menu. Change it to **true** to display the **Search** modal in the sidebar.
2. **settings**: Hides the **Settings** button in the sidebar menu. Change it to **true** to display the **Settings** button in the sidebar.
3. **default.main-menu-items**: Hides the **Create** button from the sidebar menu. Remove this field to display the **Create** button in the sidebar.

4 **position**: Defines the position of the header. Options: **above-main-content** or **above-sidebar**.

To extend the functionality of the default global header, include any the following attributes in your global header entry:

mountPoint

Specifies the location of the header. Use **application/header** to specify it as a global header. You can configure several global headers at different positions by adding entries to the **mountPoints** field.

importName

Specifies the component exported by the global header plugin.
The **red-hat-developer-hub.backstage-plugin-global-header** package (enabled by default) offers the following header components as possible mount point values:

- **SearchComponent**: Adds a search bar (enabled by default).
- **Spacer**: Adds spacing in the header to position buttons at the end. Useful when you disable **SearchComponent**.
- **HeaderIconButton**: Adds an icon button. By default, the **Create** icon button remains enabled.
- **SupportButton**: Adds a **Support** icon button, allowing users to configure a link to an internal or external page. Enabled by default but requires additional configuration to display.
- **NotificationButton**: Adds a **Notifications** icon button to display unread notifications in real time and navigate to the **Notifications** page. Enabled by default (requires the notifications plugin).
- **Divider**: Adds a vertical divider. By default, a divider appears between the profile dropdown and other header components.
- **ProfileDropdown**: Adds a profile dropdown showing the logged-in user's name. By default, it contains two menu items.
- **MenuItemLink**: Adds a link item in a dropdown menu. By default, the profile dropdown includes a link to the **Settings** page.
- **LogoutButton**: Adds a logout button in the profile dropdown (enabled by default).
- **CreateDropdown**: Adds a **Create** dropdown button (disabled by default). The menu items are configurable.
- **SoftwareTemplatesSection**: Adds a list of software template links to the **Create** dropdown menu (disabled by default). You must enable **CreateDropdown**.
- **RegisterAComponentSection**: Adds a link to the **Register a Component** page in the **Create** dropdown menu (disabled by default). You must enable **CreateDropdown**.

config.position

Specifies the position of the header. Supported values are **above-main-content** and **above-sidebar**.

Prerequisites

- You must configure the support URL in the **app-config.yaml** file to display the **Support** button in the header.
- You must install the notifications plugin to display the **Notifications** button in the header.

Procedure

1. Copy the default configuration and modify the field values to suit your needs. You can adjust the **priority** value of each header component to control its position. Additionally, you can enable or disable components by adding or removing them from the configuration. To ensure that the remaining header buttons align with the end of the header before the profile dropdown button, set **config.props.growFactor** to **1** in the **Spacer** mount point to enable the **Spacer** component. For example:

```
- mountPoint: global.header/component
  importName: Spacer
  config:
    priority: 100
    props:
      growFactor: 1
```

2. To use your custom header, you must install it as a dynamic plugin by adding your plugin configuration to your **app-config-dynamic.yaml** file. For example:

```
- package: <npm_or_oci_package-reference>
  disabled: false
  pluginConfig:
    dynamicPlugins:
      frontend:
        <package_name>:
          mountPoints:
            - mountPoint: application/header
              importName: <application_header_name>
              config:
                position: above-main-content
            - mountPoint: global.header/component
              importName: <header_component_name>
              config:
                priority: 100
            - mountPoint: global.header/component
              importName: <header_component_name>
              config:
                priority: 90
```

where:

<npm_or_oci_package-reference>

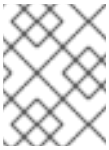
Specifies the package name.

<application_header_name>

Specifies the name of the application header. For example: **MyHeader**

<header_component_name>

Specifies the name of the header component. For example: **MyHeaderComponent**



NOTE

importName is an optional name referencing the value returned by the scaffolder field extension API.

3. Optional: To disable the global header, set the value of the **disabled** field to **true** in your **dynamic-plugins.yaml** file. For example:

```
- package: ./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-global-header
  disabled: true
```

6.2. MOUNT POINTS FOR DYNAMIC PLUGIN INTEGRATION

You can customize the application header in Developer Hub using mount points for dynamic plugins. These mount points give flexibility in configuring the position of the header, its components and dropdown menus. You can create a customized experience with the following enhancements:

application/header

Controls the header position. Use **config.position** to set placement as either **above-main-content** or **above-sidebar**.

global.header/component

Configures header components. Use **config.priority** to set the order of components, and pass properties (including CSS styles) via **config.props**.

Example adding a Self-service button

```
- mountPoint: global.header/component
  importName: HeaderIconButton
  config:
    priority: 80
    props:
      title: Self-service
      icon: add
      to: create
```

Example adding a spacer element

```
- mountPoint: global.header/component
  importName: Spacer
  config:
    priority: 99
    props:
      growFactor: 0
```

Example adding a divider element

```
mountPoints:
  - mountPoint: global.header/component
    importName: Divider
    config:
      priority: 150
```

global.header/profile

Configures the profile dropdown list when the **ProfileDropdown** component is enabled.

Example adding a settings link to the profile dropdown

```
- mountPoint: global.header/profile
  importName: MenuItemLink
  config:
    priority: 100
    props:
      title: Settings
      link: /settings
      icon: manageAccounts
```

global.header/create

Configures the create dropdown list when the **CreateDropdown** component is enabled.

Example adding a section for registering a component

```
- mountPoint: global.header/create
  importName: RegisterAComponentSection
  config:
    props:
      growFactor: 0
```

CHAPTER 7. CONFIGURING A FLOATING ACTION BUTTON IN RED HAT DEVELOPER HUB

You can use the floating action button plugin to configure any action as a floating button in the Developer Hub instance. The floating action button plugin is enabled by default. You can also configure floating action buttons to display as submenu options within the main floating action button by assigning the floating action buttons to the same **slot** field of your **dynamic-plugins.yaml** file.

7.1. CONFIGURING A FLOATING ACTION BUTTON AS A DYNAMIC PLUGIN

You can configure the floating action button as a dynamic plugin to perform actions or open an internal or external link.

Prerequisties

You must have sufficient permissions as a platform engineer.

Procedure

To configure a floating action button as a dynamic plugin, complete any of the following tasks:

- Specify the **global.floatingactionbutton/config** mount point in your **app-config-dynamic.yaml** file. For example:

  Example of a bulk-import plugin as a floating action button

```
- package: ./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import
  disabled: false
  pluginConfig:
    dynamicPlugins:
      frontend:
        red-hat-developer-hub.backstage-plugin-bulk-import:
          # Start of the floating action button configuration
          mountPoints:
            - mountPoint: global.floatingactionbutton/config
              importName: BulkImportPage (1)
              config:
                slot: 'page-end'
                icon: <svg xmlns="http://www.w3.org/2000/svg" enable-background="new 0 0 24
24" height="24px" viewBox="0 0 24 24" width="24px" fill="#e8eaed"><g><rect fill="none"
height="24" width="24"/></g><g><path d="M11,7L9.6,8.4l2.6,2.6H2v2h10.2l-2.6,2.6L11,17l5-
5L11,7z M20,19h-8v2h8c1.1,0,2-0.9,2-2V5c0-1.1-0.9-2-2-2h-8v2h8V19z"/></g></svg> (2)
                label: 'Bulk import'
                toolTip: 'Register multiple repositories in bulk'
                to: /bulk-import/repositories
          # End of the floating action button configuration
          appIcons:
            - name: bulkImportIcon
              importName: BulkImportIcon
          dynamicRoutes:
            - path: /bulk-import/repositories
              importName: BulkImportPage
```

```
        menuItem:
          icon: bulkImportIcon
          text: Bulk import
```

1. (Required) The import name with an associated component to the mount point.
2. Use the **svg** value to display a black BulkImportPage icon.

- To configure an action as a floating action button that opens an external link, specify the **global.floatingactionbutton/config** mount point in your **dynamic-plugins.yaml** file within the **backstage-plugin-global-floating-action-button** plugin. For example:

  Example of a floating action button that opens GitHub

```
- package: ./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-global-floating-action-button
  disabled: false
  pluginConfig:
    dynamicPlugins:
      frontend:
        red-hat-developer-hub.backstage-plugin-global-floating-action-button:
          mountPoints:
            - mountPoint: application/listener
              importName: DynamicGlobalFloatingActionButton
            - mountPoint: global.floatingactionbutton/config
              importName: NullComponent (1)
              config:
                icon: '<svg viewBox="0 0 250 300" xmlns="http://www.w3.org/2000/svg" preserveAspectRatio="xMidYMid"><path d="M200.134 0l55.555 117.514-55.555 117.518h-47.295l55.555-117.518L152.84 0h47.295zM110.08 99.836l20.056-38.092-2.29-8.868L102.847 0H55.552l48.647 102.898 5.881-3.062zm17.766 74.433l-17.333-39.034-6.314-3.101-48.647 102.898h47.295l25-52.88v-7.883z" fill="#40B4E5"/><path d="M152.842 235.032L97.287 117.514 152.842 0h47.295l-55.555 117.514 55.555 117.518h-47.295zm-97.287 0L0 117.514 55.555 0h47.296L47.295 117.514l55.556 117.518H55.555z" fill="#003764"/></svg>' (2)
                label: 'Quay'
                showLabel: true
                toolTip: 'Quay'
                to: 'https://quay.io'
            - mountPoint: global.floatingactionbutton/config
              importName: NullComponent
              config:
                icon: github
                label: 'Git'
                toolTip: 'Github'
                to: https://github.com/redhat-developer/rhdh-plugins
```

1. (Required) The import name with an associated component to the mount point.
2. Use the **svg** value to display the **Quay** icon.

- To configure a floating action button that contains a submenu, define the **global.floatingactionbutton/config** mount point in the same **slot** field in your **dynamic-**

plugins.yaml file for multiple actions. The default slot is **page-end** when not specified. For example:

Example of a floating action button with submenu actions

```
- package: ./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import
  disabled: false
  pluginConfig:
    dynamicPlugins:
      frontend:
        red-hat-developer-hub.backstage-plugin-bulk-import:
          # Start of fab config
          mountPoints:
            - mountPoint: global.floatingactionbutton/config
              importName: BulkImportPage (1)
              config:
                slot: 'page-end'
                icon: <svg xmlns="http://www.w3.org/2000/svg" enable-background="new 0 0 24
24" height="24px" viewBox="0 0 24 24" width="24px" fill="#e8eaed"><g><rect fill="none"
height="24" width="24"/></g><g><path d="M11,7L9.6,8.4l2.6,2.6H2v2h10.2l-2.6,2.6L11,17l5-
5L11,7z M20,19h-8v2h8c1.1,0,2-0.9,2-2V5c0-1.1-0.9-2-2-2h-8v2h8V19z"/></g></svg>
                label: 'Bulk import'
                toolTip: 'Register multiple repositories in bulk'
                to: /bulk-import/repositories
          # end of fab config
          appIcons:
            - name: bulkImportIcon
              importName: BulkImportIcon
          dynamicRoutes:
            - path: /bulk-import/repositories
              importName: BulkImportPage
              menuItem:
                icon: bulkImportIcon
                text: Bulk import

- package: ./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-global-floating-
action-button
  disabled: false
  pluginConfig:
    dynamicPlugins:
      frontend:
        red-hat-developer-hub.backstage-plugin-global-floating-action-button:
          mountPoints:
            - mountPoint: application/listener
              importName: DynamicGlobalFloatingActionButton
            - mountPoint: global.floatingactionbutton/config
              importName: NullComponent (2)
              config:
                icon: github
                label: 'Git'
                toolTip: 'Github'
                to: https://github.com/redhat-developer/rhdh-plugins
            - mountPoint: global.floatingactionbutton/config
              importName: NullComponent (3)
              config:
```

```
            icon: '<svg viewBox="0 0 250 300" xmlns="http://www.w3.org/2000/svg"
preserveAspectRatio="xMidYMid"><path d="M200.134 0l55.555 117.514-55.555 117.518h-
47.295l55.555-117.518L152.84 0h47.295zM110.08 99.836l20.056-38.092-2.29-
8.868L102.847 0H55.552l48.647 102.898 5.881-3.062zm17.766 74.433l-17.333-39.034-
6.314-3.101-48.647 102.898h47.295l25-52.88v-7.883z" fill="#40B4E5"/><path d="M152.842
235.032L97.287 117.514 152.842 0h47.295l-55.555 117.514 55.555 117.518h-47.295zm-
97.287 0L0 117.514 55.555 0h47.296L47.295 117.514l55.556 117.518H55.555z"
fill="#003764"/></svg>'
            label: 'Quay'
            showLabel: true
            toolTip: 'Quay'
            to: 'https://quay.io'
```

1 2 3 (Required) The import name with an associated component to the mount point.

- To configure a floating action button to display only on specific pages, configure the **global.floatingactionbutton/config** mount point in the **backstage-plugin-global-floating-action-button** plugin and set the **visibleOnPaths** property as shown in the following example:

Example of a floating action button to display specific pages

```
- package: ./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import
  disabled: false
  pluginConfig:
    dynamicPlugins:
      frontend:
        red-hat-developer-hub.backstage-plugin-bulk-import:
          # start of fab config
          mountPoints:
            - mountPoint: global.floatingactionbutton/config
              importName: BulkImportPage 1
              config:
                slot: 'page-end'
                icon: <svg xmlns="http://www.w3.org/2000/svg" enable-background="new 0 0 24
24" height="24px" viewBox="0 0 24 24" width="24px" fill="#e8eaed"><g><rect fill="none"
height="24" width="24"/></g><g><path d="M11,7L9.6,8.4l2.6,2.6H2v2h10.2l-2.6,2.6L11,17l5-
5L11,7z M20,19h-8v2h8c1.1,0,2-0.9,2-2V5c0-1.1-0.9-2-2-2h-8v2h8V19z"/></g></svg>
                label: 'Bulk import'
                toolTip: 'Register multiple repositories in bulk'
                to: /bulk-import/repositories
                visibleOnPaths: ['/catalog', '/settings']
          # end of fab config
          appIcons:
            - name: bulkImportIcon
              importName: BulkImportIcon
          dynamicRoutes:
            - path: /bulk-import/repositories
              importName: BulkImportPage
              menuItem:
                icon: bulkImportIcon
                text: Bulk import
```

1 (Required) The import name with an associated component to the mount point.

- To hide a floating action button on specific pages, configure the **global.floatingactionbutton/config** mount point in the **backstage-plugin-global-floating-action-button** plugin and set the **excludeOnPaths** property as shown in the following example:

Example of a floating action button to hide specific pages

```
- package: ./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import
  disabled: false
  pluginConfig:
    dynamicPlugins:
      frontend:
        red-hat-developer-hub.backstage-plugin-bulk-import:
          # start of fab config
          mountPoints:
            - mountPoint: global.floatingactionbutton/config
              importName: BulkImportPage (1)
              config:
                slot: 'page-end'
                icon: <svg xmlns="http://www.w3.org/2000/svg" enable-background="new 0 0 24
24" height="24px" viewBox="0 0 24 24" width="24px" fill="#e8eaed"><g><rect fill="none"
height="24" width="24"/></g><g><path d="M11,7L9.6,8.4l2.6,2.6H2v2h10.2l-2.6,2.6L11,17l5-
5L11,7z M20,19h-8v2h8c1.1,0,2-0.9,2-2V5c0-1.1-0.9-2-2-2h-8v2h8V19z"/></g></svg>
                label: 'Bulk import'
                toolTip: 'Register multiple repositories in bulk'
                to: /bulk-import/repositories
                excludeOnPaths: ['/bulk-import']
          # end of fab config
          appIcons:
            - name: bulkImportIcon
              importName: BulkImportIcon
          dynamicRoutes:
            - path: /bulk-import/repositories
              importName: BulkImportPage
              menuItem:
                icon: bulkImportIcon
                text: Bulk import
```

(1) (Required) The import name with an associated component to the mount point.

7.2. FLOATING ACTION BUTTON PARAMETERS

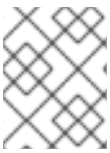
Use the parameters as shown in the following table to configure your floating action button plugin.

Table 7.1. Floating action button parameters

Name	Description	Type	Default value	Required
slot	Position of the floating action button. Valid values: **PAGE_END**, **BOTTOM_LEFT**	**enum**	**PAGE_END**	No

Name	Description	Type	Default value	Required
label	Name of the floating action button	**String**	Not applicable	Yes
icon	Icon of the floating action button. Recommended to use filled icons from the Material Design library. You can also use an svg icon. For example: **<svg xmlns="http://www.w3.org/2000/svg" enable-background="new 0 0 24 24" height="24px" viewBox="0 0 24 24" width="24px" fill="#e8eaed"><g><rect fill="none" height="24" width="24"/></g><g><path d="M11,7L9.6,8.4l2.6,2.6H2v2h10.2l-2.6,2.6L11,17l5-5L11,7z M20,19h-8v2h8c1.1,0,2-0.9,2-2V5c0-1.1-0.9-2-2-2h-8v2h8V19z"/></g></svg>**	**String**, **React.ReactElement**, **SVG image icon**, **HTML image icon**	Not applicable	No
showLabel	Display of the label next to your icon	**Boolean**	Not applicable	No
size	Size of the floating action button	**small**, **medium**, **large**	**medium**	No

Name	Description	Type	Default value	Required
color	Color of the component. It supports both default and custom theme colors, that are added from the Palette Getting started guide.	**default**, **error**, **info**, **inherit**, **primary**, **secondary**, **success**, **warning**	**default**	No
onClick	Performed action when selecting a floating action button	**React.MouseEventHandler**	Not applicable	No
to	Link that opens when selecting a floating action button	**String**	Not applicable	No
toolTip	Text that appears when hovering over a floating action button	**String**	Not applicable	No
priority	Order of the floating action buttons displayed in the submenu. A larger value means higher priority.	**number**	Not applicable	No
visibleOnPaths	Display floating action button on the specified paths	**string[]**	Display floating action button on all paths	No
excludeOnPaths	Hide floating action button on the specified paths	**string[]**	Display floating action button on all paths	No



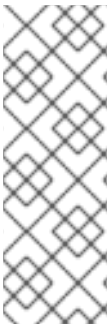
NOTE

If multiple floating button actions are assigned to the same **slot** value, the floating buttons are displayed as submenu options within the main floating action button.

CHAPTER 8. CUSTOMIZING THE TECH RADAR PAGE IN RED HAT DEVELOPER HUB

In Red Hat Developer Hub, the Tech Radar page is provided by the **tech-radar** dynamic plugin, which is disabled by default. For information about enabling dynamic plugins in Red Hat Developer Hub see Configuring dynamic plugins.

In Red Hat Developer Hub, you can configure Learning Paths by passing the data into the **app-config.yaml** file as a proxy. The base Tech Radar URL must include the **/developer-hub/tech-radar** proxy.



NOTE

Due to the use of overlapping **pathRewrites** for both the **tech-radar** and **homepage** quick access proxies, you must create the **tech-radar** configuration (**^api/proxy/developer-hub/tech-radar**) before you create the **homepage** configuration (**^/api/proxy/developer-hub**).

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

You can provide data to the Tech Radar page from the following sources:

- JSON files hosted on GitHub or GitLab.
- A dedicated service that provides the Tech Radar data in JSON format using an API.

8.1. CUSTOMIZING THE TECH RADAR PAGE BY USING A JSON FILE

For ease of use and simplicity, you can configure the Tech Radar page by using a hosted JSON file.

Prerequisites

- You have specified the data sources for the Tech Radar plugin in the **integrations** section of the **app-config.yaml** file. For example, to configure GitHub as an integration, see Authenticating with GitHub.
- You have enabled the **./dynamic-plugins/dist/backstage-community-plugin-tech-radar** and **/dynamic-plugins/dist/backstage-community-plugin-tech-radar-backend-dynamic** plugins.

Procedure

1. Publish the JSON file containing your Tech Radar data to a web server, such as GitHub or Gitlab. You can find an example at https://raw.githubusercontent.com/redhat-developer/rhdh/release-1.5/packages/app/public/tech-radar/data-default.json.
2. Configure Developer Hub to access the Tech Radar data from the hosted JSON files, by adding the following to the **app-config.yaml** file:

```
techRadar:
  url: <tech_radar_data_url>
```

<tech_radar_data_url>

Enter the Tech Radar data hosted JSON URL.

8.2. CUSTOMIZING THE TECH RADAR PAGE BY USING A CUSTOMIZATION SERVICE

For advanced scenarios, you can host your Red Hat Developer Hub customization service to provide data to all configurable Developer Hub pages, such as the Tech Radar page. You can even use a different service for each page.

Prerequisites

- You have specified the data sources for the Tech Radar plugin in the **integrations** section of the **app-config.yaml** file. For example, to configure GitHub as an integration, see Authenticating with GitHub.
- You have enabled the **./dynamic-plugins/dist/backstage-community-plugin-tech-radar** and **/dynamic-plugins/dist/backstage-community-plugin-tech-radar-backend-dynamic** plugins.

Procedure

1. Deploy your Developer Hub customization service on the same OpenShift Container Platform cluster as your Developer Hub instance. You can find an example at **red-hat-developer-hub-customization-provider**, that provides the same data as default Developer Hub data. The customization service provides a Tech Radar data URL such as: **http://*<rhdh-customization-provider>*/tech-radar**.
2. Add the dedicated service as an allowed host by adding the following code to the **app-config.yaml** file:

```
backend:
  reading:
    allow:
      - host: '<rhdh_customization_provider_base_url>'
```

<rhdh_customization_provider_base_url>

Enter the base URL of your Tech Radar data URL, such as: ***<rhdh-customization-provider>***.

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

```
techRadar:
  url: <tech_radar_data_url>
```

<tech_radar_data_url>

Enter your Tech Radar data URL, such as: **http://*<rhdh-customization-provider>*/tech-radar**.

CHAPTER 9. CUSTOMIZING RED HAT DEVELOPER HUB APPEARANCE

The following default theme configurations are available for Red Hat Developer Hub:

The Red Hat Developer Hub theme

Default theme configurations to make your developer portal look like a standard Red Hat Developer Hub instance. For more information, see Section 9.9, "Default Red Hat Developer Hub theme"

The Backstage theme

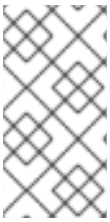
Default theme configurations to make your developer portal look like a standard Backstage instance. For more information, see Section 9.10, "Default Backstage theme"

You can change or disable particular parameters in a default theme or create a fully customized theme by modifying the **app-config-rhdh.yaml** file. From the **app-config-rhdh.yaml** file, you can customize common theme components, including the following:

- Company name and logo
- Font color, size, and style of text in paragraphs, headings, headers, and buttons
- Header color, gradient, and shape
- Button color
- Navigation indicator color

You can also customize some components from the Developer Hub GUI, such as the theme mode (**Light Theme**, **Dark Theme**, or **Auto**).

9.1. CUSTOMIZING THE THEME MODE FOR YOUR DEVELOPER HUB INSTANCE



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.

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

- Light theme
- Dark theme
- Auto

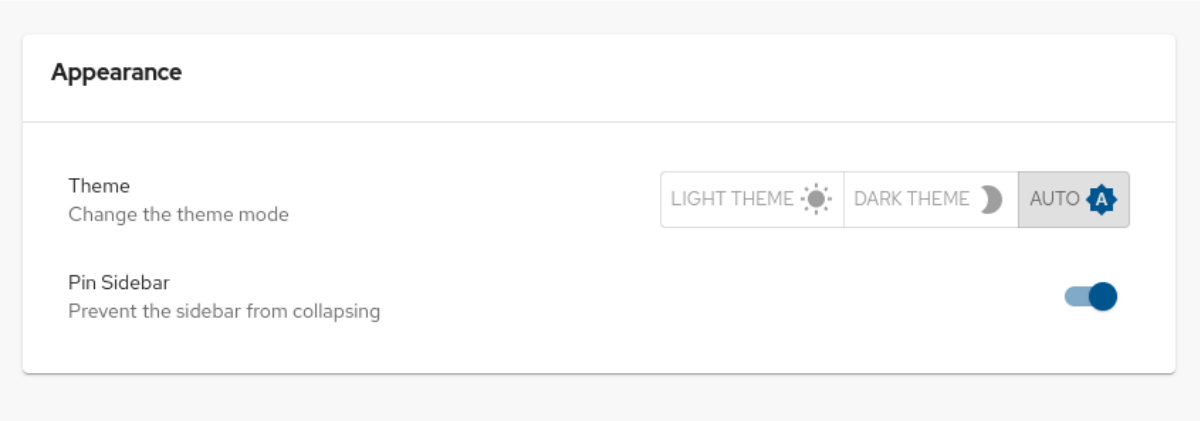
The default theme mode is Auto, which automatically sets the light or dark theme based on your system preferences.

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 THEME**, **DARK THEME**, or **AUTO** to change the theme mode.



9.2. CUSTOMIZING THE BRANDING LOGO OF YOUR DEVELOPER HUB INSTANCE

You can customize the branding logo of your Developer Hub instance by configuring the **branding** section in the **app-config.yaml** file, as shown in the following example:

```
app:
  branding:
    fullLogo: ${BASE64_EMBEDDED_FULL_LOGO} 1
    iconLogo: ${BASE64_EMBEDDED_ICON_LOGO} 2
```

where:

1. **fullLogo** is the logo on the expanded (pinned) sidebar and expects a base64 encoded image.
2. **iconLogo** is the logo on the collapsed (unpinned) sidebar and expects a base64 encoded image.

You can format the **BASE64_EMBEDDED_FULL_LOGO** environment variable as follows:

```
BASE64_EMBEDDED_FULL_LOGO: "data:_<media_type>_;base64,<base64_data>"
```

The following example demonstrates how to customize the **BASE64_EMBEDDED_FULL_LOGO** using the **data:_<media_type>_;base64,*<base64_data>*** format:

```
SVGLOGOBASE64=$(base64 -i logo.svg)
BASE64_EMBEDDED_FULL_LOGO="$SVGLOGOBASE64"
```

Replace **image/svg+xml** with the correct media type for your image (for example, **image/png** and **image/jpeg**), and adjust the file extension accordingly. As a result, you can embed the logo directly without referencing an external file.

You can also customize the width of the branding logo by setting a value for the **fullLogoWidth** field in the **branding** section, as shown in the following example:

```
app:
  branding:
    fullLogoWidth: 110px (1)
# ...
```

(1) The default value for the logo width is **110px**. The following units are supported: **integer**, **px**, **em**, **rem**, percentage.

9.3. CUSTOMIZING THE SIDEBAR MENU ITEMS FOR YOUR DEVELOPER HUB INSTANCE

The sidebar menu in Red Hat Developer Hub consists of two main parts that you can configure:

Dynamic plugin menu items

Your preferences and your active plugins define dynamically one part of the sidebar menu.

Main menu items

The core navigation structure of sidebar is static.

- **Dynamic plugin menu items**: These items are displayed beneath the main menu and can be customized based on the plugins installed. The main menu items section is dynamic and can change based on your preferences and installed plugins.

9.3.1. Customizing the sidebar menu items for your Developer Hub instance

Customize the main menu items using the following steps:

Procedure

1. Open the **app-config.yaml** file.
   a. To customize the order and parent-child relationships for the main menu items, use the **dynamicPlugins.frontend.default.main-menu-items.menuItems** field.
   b. For dynamic plugin menu items, use the **dynamicPlugins.frontend.<package_name>.menuItems** field.

Example app-config.yaml file

```
dynamicPlugins:
  frontend:
    default.main-menu-items:
      menuItems:
        default.home:
          title: Home
          icon: home
          priority: 100
        default.my-group:
          title: My Group
          icon: group
          priority: 90
        default.catalog:
```

```
    title: Catalog
    icon: category
    to: catalog
    priority: 80
  default.apis:
    title: APIs
    icon: extension
    to: api-docs
    priority: 70
  default.learning-path:
    title: Learning Paths
    icon: school,
    to: learning-paths
    priority: 60
  default.create:
    title: Create
    icon: add
    to: create
    priority: 50
```

9.3.2. Configuring a dynamic plugin menu item for your Developer Hub instance

Configure a dynamic plugin menu item using the following step:

Procedure

- In the **app-config.yaml** file, update the **menuItems** section of your *<plugin_name>* plugin. For example:

```
dynamicPlugins:
  frontend:
    _<plugin_name>_: 1
      menuItems:
        <menu_item_name>: 2
          icon: # home | group | category | extension | school | _<my_icon>_ 3
          title: _<plugin_page_title>_ 4
          priority: 10 5
          parent: favorites 6
```

1. ***<plugin_name>***: Enter the plugin name. This name is the same as the **scalprum.name** key in the **package.json** file.

2. ***<menu_item_name>***: Enter a unique name in the main sidebar navigation for either a standalone menu item or a parent menu item. If this field specifies a plugin menu item, the name of the menu item must match the name using in the corresponding path in **dynamicRoutes**. For example, if **dynamicRoutes** defines **path: /my-plugin**, then **menu_item_name** must be defined as **my-plugin**.

3. **icon**: (Optional) Enter the icon name. You can use any of the following icons:

    - Default icons, such as **home**, **group**, **category**, **extension**, and **school**. To use default icons, set the icon as an (**" "**) empty string.

    - A custom icon, where *<my_icon>* specifies the name of your custom icon

- An SVG icon, such as: **icon: <svg width="20px" height="20px" viewBox="0 0 512 512" xmlns="http://www.w3.org/2000/svg" fill="#ffffff">…</svg>**
- An HTML image, such as: **icon: https://img.icons8.com/ios-glyphs/20/FFFFFF/shop.png**

4. **title**: (Optional) Enter the menu item title. Omit it when the title is already specified in the **dynamicRoutes** configuration under **menuItem.text**. To hide the title from the sidebar, set the title as an (**" "**) empty string.
5. **priority**: (Optional) Sets the order in which menu items appear in the sidebar. The default priority is 0, which places the item at the bottom of the list. A higher priority value places the item higher in the sidebar. You can define this field for each section.
6. **parent**: (Optional) Enter the parent menu item under which the current item is nested. If this field is used, the parent menu item must be defined elsewhere in the **menuItems** configuration of any enabled plugin. You can define this field for each section.

Example **menuItems** configuration

```
dynamicPlugins:
  frontend:
    _<package_name>_:
      dynamicRoutes:
        - path: /my-plugin
          module: CustomModule
          importName: FooPluginPage
          menuItem:
            icon: fooIcon
            text: Foo Plugin Page
      menuItems:
        my-plugin: 1
          priority: 10 2
          parent: favorites 3
        favorites: 4
          icon: favorite 5
          title: Favorites 6
          priority: 100 7
```

1. **my-plugin**: Matches the value of the **path** field in **dynamicRoutes**.
2. **priority**: Controls order of plugins under the parent menu item.
3. **parent**: Nests this plugin under the **favorites** parent menu item.
4. **favorites**: Configuration for the parent menu item.
5. **icon**: Displays the **favorite** icon from the RHDH system icons.
6. **title**: Displays the title name for the parent menu item.
7. **priority**: Order of the **favourites** menu item in the sidebar.

9.3.3. Modifying or adding a custom menu items for your Developer Hub instance

Modify a main menu item or add a custom menu item using the following step:

Procedure

- In the **app-config.yaml** file, add a section to the **default.main-menu-items** > **menuItems** section. Use the **default.** prefix to identify the key as a main menu item.

```
dynamicPlugins:
  frontend:
    default.main-menu-items:
      menuItems:
        default._<menu_group_parent_item_name>_: 1
          icon: # home | group | category | extension | school | _<my_icon>_ 2
          title: _<menu_group_parent_title>_ 3
          priority: 10 4
        default._<menu_item_name>_: 5
          parent: _<menu_group_parent_item_name>_ 6
          icon:  # home | group | category | extension | school | _<my_icon>_ 7
          title: _<my_menu_title>_ 8
          to: _<path_to_the_menu_target_page>_ 9
          priority: 100 10
```

1. **default.*<menu_group_parent_item_name>***: (Optional) Enter the menu group parent item name to configure static main menu items. If no **default.*<menu_item_name>*** has a **parent** value set, this field is not needed.
2. **icon**: Enter the menu icon. Required for parent menu items.
3. **title**: Enter the menu group title. Required for parent menu items.
4. **priority**: (Optional) Enter the order of this menu item within its menu level.
5. **default.*<menu_item_name>***: Enter the menu item name for which you want to override the default value. Add the **default.** prefix to identify a main menu item.
6. **parent**: (Optional) Enter the parent menu item for this item. Required if <menu_item_name> is specified as the child of any menu items.
7. **icon**: (Optional) Enter the menu icon. To use the default icon, set the icon as an (" ") empty string.
8. **title**: (Optional) Enter the menu group title. Only required for adding a new custom main menu item. To hide a default main menu item title from the sidebar, set the title as an (" ") empty string.
9. **to**: (Optional) Enter the path that the menu item navigates to. If it is not set, it defaults to the home page.
10. **priority**: (Optional) Enter the order of this menu item within its menu level.

Example mainItems configuration

```
default.main-menu-items:
    menuItems:
      default.catalog:
        icon: category (1)
        title: My Catalog
        priority: 5
      default.learning-path:
        title: '' (2)
      default.parentlist: (3)
        title: Overview
        icon: bookmarks
      default.home:
        parent: default.parentlist (4)
      default.references:
        title: References (5)
        icon: school (6)
        to: /references (7)
```

1. **icon**: Specifies if you want to change the icon default menu item for the catalog.
2. **title**: Specifies an empty string **'' ''** to hide the learning path from the default sidebar.
3. **default.parentlist**: Introduces the parent menu item.
4. **parent**: Nests home menu under the **default.parentlist** parent menu item.
5. **title**: Specifies a name for **default.references**
6. **icon**: Displays the **school** icon.
7. **to**: Redirects **default.references** to the **/references** page.

9.4. CONFIGURING ENTITY TAB TITLES

Red Hat Developer Hub provides a default opinionated tab set for catalog entity views. For consistency with your organization needs, you can rename, reorder, remove, and add tab titles.

Procedure

- For each tab to modify, enter your desired values in the **entityTabs** section in your **app-config.yaml** file:

```
upstream:
  backstage:
    appConfig:
      dynamicPlugins:
        frontend:
          <plugin_name>:
            entityTabs:
              - mountPoint: <mount_point>
                path: <path>
                title: <title>
                priority: <priority>
```

<plugin_name>

Enter the plugin name, such as **backstage-community.plugin-topology**.

mountPoint

Enter the tab mountpoint, such as **entity.page.topology**.

path

Enter the tab path, such as /**topology**.

title

Enter the tab title, such as **Topology**.

priority

Optional.
To reorder tabs, enter the tab priority, such as **42**. Higher priority appears first.

To remove a tab, enter a negative value, such as **-1**.

9.5. CONFIGURING ENTITY DETAIL TAB LAYOUT

Each Red Hat Developer Hub entity detail tab has a default opinionated layout. For consistency with your organization needs, you can change the entity detail tab content when the plugin that contributes the tab content allows a configuration.

Prerequisites

- The plugin that contributes the tab content allows a configuration, such as Developer Hub plugins defining a default configuration in a **config** section.

Procedure

- Copy the plugin default configuration in your **app-config.yaml** file, and change the **layout** properties.

```
global:
  dynamic:
    plugins:
      - package: <package_location>
        disabled: false
        pluginConfig:
          dynamicPlugins:
            frontend:
              <plugin_name>:
                mountPoints:
                  - mountPoint: <mount_point>
                    importName: <import_name>
                    config:
                      layout:
                        gridColumn:
                          lg: span 6
                          xs: span 12
```

package

Enter your package location, such as **./dynamic-plugins/dist/backstage-community-plugin-tekton**.

<plugin_name>

Enter your plugin name, such as: **backstage-community.plugin-tekton**.

mountPoint

Copy the mount point defined in the plugin default configuration, such as: **entity.page.ci/cards**.

importName

Copy the import name defined in the plugin default configuration, such as: **TektonCI**.

layout

Enter your layout configuration. The tab content is displayed in a responsive grid that uses a 12 column-grid and supports different breakpoints (**xs**, **sm**, **md**, **lg**, **xl**) that can be specified for a CSS property, such as **gridColumn**. The example uses 6 of the 12 columns to show two Tekton CI cards side-by-side on large (**lg**) screens (**span 6** columns) and show them among themselves (**xs** and above **span 12** columns).

9.6. CUSTOMIZING THE THEME MODE COLOR PALETTES FOR YOUR DEVELOPER HUB INSTANCE

You can customize the color palettes of the light and dark theme modes in your Developer Hub instance by configuring the **light.palette** and **dark.palette** parameters in the **branding.theme** section of the **app-config.yaml** file, as shown in the following example:

```
app:
  branding:
    theme:
      light:
        palette:
          primary:
            main: <light_primary_color> 1
          navigation:
            indicator: <light_indicator_color> 2
        pageTheme:
          default:
            backgroundColor: [<light_background_color_1>, <light_background_color_2>] 3
      dark:
        palette:
          primary:
            main: <dark_primary_color> 4
          navigation:
            indicator: <dark_indicator_color> 5
        pageTheme:
          default:
            backgroundColor: [<dark_background_color_1>, <dark_background_color_2>] 6
# ...
```

1. The main primary color for the light color palette, for example, **#ffffff** or **white**
2. The color of the navigation indicator for the light color palette, which is a vertical bar that indicates the selected tab in the navigation panel, for example, **#FF0000** or **red**

3. The background color for the default page theme for the light color palette, for example, **#ffffff** or **white**
4. The main primary color for the dark color palette, for example, **#000000** or **black**
5. The color of the navigation indicator for the dark color palette, which is a vertical bar that indicates the selected tab in the navigation panel, for example, **#FF0000** or **red**
6. The background color for the default page theme for the dark color palette, for example, **#000000** or **black**

Additional resources

- Section 9.1, "Customizing the theme mode for your Developer Hub instance"

9.7. CUSTOMIZING THE PAGE THEME HEADER FOR YOUR DEVELOPER HUB INSTANCE

You can customize the header color for the light and dark theme modes in your Developer Hub instance by modifying the **branding.theme** section of the **app-config.yaml** file. You can also customize the page headers for additional Developer Hub pages, such as the **Home**, **Catalog**, and **APIs** pages.

```
app:
  branding:
    theme:
      light: 1
        palette: {}
        pageTheme:
          default: 2
            backgroundColor: "<default_light_background_color>" 3
            fontColor: "<default_light_font_color>" 4
            shape: none 5
          apis: 6
            backgroundColor: "<apis_light_background_color>"
            fontColor: "<apis_light_font_color>"
            shape: none
      dark:
        palette: {}
        pageTheme:
          default:
            backgroundColor: "<default_dark_background_color>"
            fontColor: "<default_dark_font_color>"
            shape: none
# ...
```

1. The theme mode, for example, **light** or **dark**
2. The **yaml** header for the default page theme configuration
3. The color of the page header background, for example, **#ffffff** or **white**
4. The color of the text in the page header, for example, **#000000** or **black**

5 The pattern on the page header, for example, **wave**, **round**, or **none**

6 The **yaml** header for a specific page theme configuration, for example, **apis**, **home**

9.8. CUSTOMIZING THE FONT FOR YOUR DEVELOPER HUB INSTANCE

You can configure the **typography** section of the **app-config.yaml** file to change the default font family and size of the page text, as well as the font family and size of each heading level, as shown in the following example:

```
app:
  branding:
    theme:
      light:
        typography:
          fontFamily: "Times New Roman"
          htmlFontSize: 11 # smaller is bigger
          h1:
            fontFamily: "Times New Roman"
            fontSize: 40
          h2:
            fontFamily: "Times New Roman"
            fontSize: 30
          h3:
            fontFamily: "Times New Roman"
            fontSize: 30
          h4:
            fontFamily: "Times New Roman"
            fontSize: 30
          h5:
            fontFamily: "Times New Roman"
            fontSize: 30
          h6:
            fontFamily: "Times New Roman"
            fontSize: 30
      dark:
        typography:
          fontFamily: "Times New Roman"
          htmlFontSize: 11 # smaller is bigger
          h1:
            fontFamily: "Times New Roman"
            fontSize: 40
          h2:
            fontFamily: "Times New Roman"
            fontSize: 30
          h3:
            fontFamily: "Times New Roman"
            fontSize: 30
          h4:
            fontFamily: "Times New Roman"
            fontSize: 30
          h5:
            fontFamily: "Times New Roman"
```

```
        fontSize: 30
      h6:
        fontFamily: "Times New Roman"
        fontSize: 30
# ...
```

9.9. DEFAULT RED HAT DEVELOPER HUB THEME

You can use the default Red Hat Developer Hub theme configurations to make your Developer Hub instance look like a standard Red Hat Developer Hub instance. You can also modify the **app-config.yaml** file to customize or disable particular parameters.

9.9.1. Default Red Hat Developer Hub theme color palette

The **app-config.yaml** file uses the following configurations for the default Red Hat Developer Hub color palette:

```
app:
  branding:
    theme:
      light:
        variant: "rhdh"
        mode: "light"
        palette:
          background:
            default: "#F8F8F8"
            paper: "#FFFFFF"
          banner:
            closeButtonColor: "#FFFFFF"
            error: "#E22134"
            info: "#2E77D0"
            link: "#000000"
            text: "#FFFFFF"
            warning: "#FF9800"
          border: "#E6E6E6"
          bursts:
            backgroundColor:
              default: "#7C3699"
            fontColor: "#FEFEFE"
            gradient:
              linear: "linear-gradient(-137deg, #4BB8A5 0%, #187656 100%)"
            slackChannelText: "#ddd"
          errorBackground: "#FFEBEE"
          errorText: "#CA001B"
          gold: "#FFD600"
          highlight: "#FFFBCC"
          infoBackground: "#ebf5ff"
          infoText: "#004e8a"
          link: "#0A6EBE"
          linkHover: "#2196F3"
          mode: "light"
          navigation:
            background: "#222427"
            indicator: "#0066CC"
            color: "#ffffff"
```

```
    selectedColor: "#ffffff"
    navItem:
      hoverBackground: "#3c3f42"
    submenu:
      background: "#222427"
  pinSidebarButton:
    background: "#BDBDBD"
    icon: "#181818"
  primary:
    main: "#0066CC"
  secondary:
    main: "#8476D1"
  status:
    aborted: "#757575"
    error: "#E22134"
    ok: "#1DB954"
    pending: "#FFED51"
    running: "#1F5493"
    warning: "#FF9800"
  tabbar:
    indicator: "#9BF0E1"
  textContrast: "#000000"
  textSubtle: "#6E6E6E"
  textVerySubtle: "#DDD"
  warningBackground: "#F59B23"
  warningText: "#000000"
  text:
    primary: "#151515"
    secondary: "#757575"
  rhdh:
    general:
      disabledBackground: "#D2D2D2"
      disabled: "#6A6E73"
      searchBarBorderColor: "#E4E4E4"
      formControlBackgroundColor: "#FFF"
      mainSectionBackgroundColor: "#FFF"
      headerBottomBorderColor: "#C7C7C7"
      cardBackgroundColor: "#FFF"
      sidebarBackgroundColor: "#212427"
      cardBorderColor: "#C7C7C7"
      tableTitleColor: "#181818"
      tableSubtitleColor: "#616161"
      tableColumnTitleColor: "#151515"
      tableRowHover: "#F5F5F5"
      tableBorderColor: "#E0E0E0"
      tableBackgroundColor: "#FFF"
      tabsBottomBorderColor: "#D2D2D2"
      contrastText: "#FFF"
    primary:
      main: "#0066CC"
      focusVisibleBorder: "#0066CC"
    secondary:
      main: "#8476D1"
      focusVisibleBorder: "#8476D1"
    cards:
      headerTextColor: "#151515"
```

```
        headerBackgroundColor: "#FFF"
        headerBackgroundImage: "none"

  dark:
    variant: "rhdh"
    mode: "dark"
    palette:
      background:
        default: "#333333"
        paper: "#424242"
      banner:
        closeButtonColor: "#FFFFFF"
        error: "#E22134"
        info: "#2E77D0"
        link: "#000000"
        text: "#FFFFFF"
        warning: "#FF9800"
      border: "#E6E6E6"
      bursts:
        backgroundColor:
          default: "#7C3699"
        fontColor: "#FEFEFE"
        gradient:
          linear: "linear-gradient(-137deg, #4BB8A5 0%, #187656 100%)"
        slackChannelText: "#ddd"
      errorBackground: "#FFEBEE"
      errorText: "#CA001B"
      gold: "#FFD600"
      highlight: "#FFFBCC"
      infoBackground: "#ebf5ff"
      infoText: "#004e8a"
      link: "#9CC9FF"
      linkHover: "#82BAFD"
      mode: "dark"
      navigation:
        background: "#0f1214"
        indicator: "#0066CC"
        color: "#ffffff"
        selectedColor: "#ffffff"
        navItem:
          hoverBackground: "#3c3f42"
        submenu:
          background: "#0f1214"
      pinSidebarButton:
        background: "#BDBDBD"
        icon: "#404040"
      primary:
        main: "#1FA7F8"
      secondary:
        main: "#B2A3FF"
      status:
        aborted: "#9E9E9E"
        error: "#F84C55"
        ok: "#71CF88"
        pending: "#FEF071"
        running: "#3488E3"
```

```
      warning: "#FFB84D"
    tabbar:
      indicator: "#9BF0E1"
    textContrast: "#FFFFFF"
    textSubtle: "#CCCCCC"
    textVerySubtle: "#727272"
    warningBackground: "#F59B23"
    warningText: "#000000"

    rhdh:
      general:
        disabledBackground: "#444548"
        disabled: "#AAABAC"
        searchBarBorderColor: "#57585a"
        formControlBackgroundColor: "#36373A"
        mainSectionBackgroundColor: "#0f1214"
        headerBottomBorderColor: "#A3A3A3"
        cardBackgroundColor: "#292929"
        sidebarBackgroundColor: "#1b1d21"
        cardBorderColor: "#A3A3A3"
        tableTitleColor: "#E0E0E0"
        tableSubtitleColor: "#E0E0E0"
        tableColumnTitleColor: "#E0E0E0"
        tableRowHover: "#0f1214"
        tableBorderColor: "#515151"
        tableBackgroundColor: "#1b1d21"
        tabsBottomBorderColor: "#444548"
        contrastText: "#FFF"
      primary:
        main: "#1FA7F8"
        focusVisibleBorder: "#ADD6FF"
      secondary:
        main: "#B2A3FF"
        focusVisibleBorder: "#D0C7FF"
      cards:
        headerTextColor: "#FFF"
        headerBackgroundColor: "#0f1214"
        headerBackgroundImage: "none"
```

Alternatively, you can use the following **variant** and **mode** values in the **app-config.yaml** file to apply the previous default configuration:

```
app:
  branding:
    theme:
      light:
        variant: "rhdh"
        mode: "light"
      dark:
        variant: "rhdh"
        mode: "dark"
```

9.9.2. Default Red Hat Developer Hub page themes

The default Developer Hub header color is white in light mode and black in dark mode, as shown in the following **app-config.yaml** file configuration:

```
app:
  branding:
    theme:
      light:
        palette: {}
        defaultPageTheme: default
        pageTheme:
          default:
            backgroundColor: "#ffffff"
      dark:
        palette: {}
        defaultPageTheme: default
        pageTheme:
          default:
            backgroundColor: "#0f1214"
```

9.10. DEFAULT BACKSTAGE THEME

You can use the default Backstage theme configurations to make your Developer Hub instance look like a standard Backstage instance. You can also modify the **app-config.yaml** file to customize or disable particular parameters.

9.10.1. Default Backstage theme color palette

The **app-config.yaml** file uses the following configurations for the default Backstage color palette:

```
app:
  branding:
    theme:
      light:
        variant: "backstage"
        mode: "light"
        palette:
          background:
            default: "#F8F8F8"
            paper: "#FFFFFF"
          banner:
            closeButtonColor: "#FFFFFF"
            error: "#E22134"
            info: "#2E77D0"
            link: "#000000"
            text: "#FFFFFF"
            warning: "#FF9800"
          border: "#E6E6E6"
          bursts:
            backgroundColor:
              default: "#7C3699"
            fontColor: "#FEFEFE"
            gradient:
              linear: "linear-gradient(-137deg, #4BB8A5 0%, #187656 100%)"
            slackChannelText: "#ddd"
```

```
      errorBackground: "#FFEBEE"
      errorText: "#CA001B"
      gold: "#FFD600"
      highlight: "#FFFBCC"
      infoBackground: "#ebf5ff"
      infoText: "#004e8a"
      link: "#0A6EBE"
      linkHover: "#2196F3"
      navigation:
        background: "#171717"
        color: "#b5b5b5"
        indicator: "#9BF0E1"
        navItem:
          hoverBackground: "#404040"
        selectedColor: "#FFF"
        submenu:
          background: "#404040"
      pinSidebarButton:
        background: "#BDBDBD"
        icon: "#181818"
      primary:
        main: "#1F5493"
      status:
        aborted: "#757575"
        error: "#E22134"
        ok: "#1DB954"
        pending: "#FFED51"
        running: "#1F5493"
        warning: "#FF9800"
      tabbar:
        indicator: "#9BF0E1"
      textContrast: "#000000"
      textSubtle: "#6E6E6E"
      textVerySubtle: "#DDD"
      warningBackground: "#F59B23"
      warningText: "#000000"

  dark:
    variant: "backstage"
    mode: "dark"
    palette:
      background:
        default: "#333333"
        paper: "#424242"
      banner:
        closeButtonColor: "#FFFFFF"
        error: "#E22134"
        info: "#2E77D0"
        link: "#000000"
        text: "#FFFFFF"
        warning: "#FF9800"
      border: "#E6E6E6"
      bursts:
        backgroundColor:
          default: "#7C3699"
        fontColor: "#FEFEFE"
```

```
        gradient:
          linear: "linear-gradient(-137deg, #4BB8A5 0%, #187656 100%)"
        slackChannelText: "#ddd"
      errorBackground: "#FFEBEE"
      errorText: "#CA001B"
      gold: "#FFD600"
      highlight: "#FFFBCC"
      infoBackground: "#ebf5ff"
      infoText: "#004e8a"
      link: "#9CC9FF"
      linkHover: "#82BAFD"
      mode: "dark"
      navigation:
        background: "#424242"
        color: "#b5b5b5"
        indicator: "#9BF0E1"
        navItem:
          hoverBackground: "#404040"
        selectedColor: "#FFF"
        submenu:
          background: "#404040"
      pinSidebarButton:
        background: "#BDBDBD"
        icon: "#404040"
      primary:
        dark: "#82BAFD"
        main: "#9CC9FF"
      secondary:
        main: "#FF88B2"
      status:
        aborted: "#9E9E9E"
        error: "#F84C55"
        ok: "#71CF88"
        pending: "#FEF071"
        running: "#3488E3"
        warning: "#FFB84D"
      tabbar:
        indicator: "#9BF0E1"
      textContrast: "#FFFFFF"
      textSubtle: "#CCCCCC"
      textVerySubtle: "#727272"
      warningBackground: "#F59B23"
      warningText: "#000000"
```

Alternatively, you can use the following **variant** and **mode** values in the **app-config.yaml** file to apply the previous default configuration:

```
app:
  branding:
    theme:
      light:
        variant: "backstage"
        mode: "light"
      dark:
        variant: "backstage"
        mode: "dark"
```

-

9.10.2. Default Backstage page themes

The default Backstage header color is white in light mode and black in dark mode, as shown in the following **app-config.yaml** file configuration:

```
app:
  branding:
    theme:
      light:
        palette: {}
        defaultPageTheme: default
        pageTheme:
          default:
            backgroundColor: ['#005B4B'] # teal
            fontColor: '#ffffff'
            shape: wave
          documentation:
            backgroundColor: ['#C8077A', '#C2297D'] # pinkSea
            fontColor: '#ffffff'
            shape: wave2
          tool:
            backgroundColor: ['#8912CA', '#3E00EA'] # purpleSky
            fontColor: '#ffffff'
            shape: round
          service:
            backgroundColor: ['#006D8F', '#0049A1'] # marineBlue
            fontColor: '#ffffff'
            shape: wave
          website:
            backgroundColor: ['#0027AF', '#270094'] # veryBlue
            fontColor: '#ffffff'
            shape: wave
          library:
            backgroundColor: ['#98002B', '#8D1134'] # rubyRed
            fontColor: '#ffffff'
            shape: wave
          other:
            backgroundColor: ['#171717', '#383838'] # darkGrey
            fontColor: '#ffffff'
            shape: wave
          app:
            backgroundColor: ['#BE2200', '#A41D00'] # toastyOrange
            fontColor: '#ffffff'
            shape: shapes.wave
          apis:
            backgroundColor: ['#005B4B'] # teal
            fontColor: '#ffffff'
            shape: wave2
          card:
            backgroundColor: ['#4BB8A5', '#187656'] # greens
            fontColor: '#ffffff'
            shape: wave

      dark:
```

```
    palette: {}
  defaultPageTheme: default
  pageTheme:
    default:
      backgroundColor: ['#005B4B'] # teal
      fontColor: '#ffffff'
      shape: wave
    documentation:
      backgroundColor: ['#C8077A', '#C2297D'] # pinkSea
      fontColor: '#ffffff'
      shape: wave2
    tool:
      backgroundColor: ['#8912CA', '#3E00EA'] # purpleSky
      fontColor: '#ffffff'
      shape: round
    service:
      backgroundColor: ['#006D8F', '#0049A1'] # marineBlue
      fontColor: '#ffffff'
      shape: wave
    website:
      backgroundColor: ['#0027AF', '#270094'] # veryBlue
      fontColor: '#ffffff'
      shape: wave
    library:
      backgroundColor: ['#98002B', '#8D1134'] # rubyRed
      fontColor: '#ffffff'
      shape: wave
    other:
      backgroundColor: ['#171717', '#383838'] # darkGrey
      fontColor: '#ffffff'
      shape: wave
    app:
      backgroundColor: ['#BE2200', '#A41D00'] # toastyOrange
      fontColor: '#ffffff'
      shape: shapes.wave
    apis:
      backgroundColor: ['#005B4B'] # teal
      fontColor: '#ffffff'
      shape: wave2
    card:
      backgroundColor: ['#4BB8A5', '#187656'] # greens
      fontColor: '#ffffff'
      shape: wave
```

9.11. LOADING A CUSTOM DEVELOPER HUB THEME BY USING A DYNAMIC PLUGIN

You can load a custom Developer Hub theme from a dynamic plugin.

Procedure

1. Export a theme provider function in your dynamic plugin, for example:

   Sample myTheme.ts fragment

   ▪

```
import { lightTheme } from './lightTheme'; // some custom theme
import { UnifiedThemeProvider } from '@backstage/theme';
export const lightThemeProvider = ({ children }: { children: ReactNode }) => (
  <UnifiedThemeProvider theme={lightTheme} children={children} />
);
```

For more information about creating a custom theme, see Backstage documentation - Creating a Custom Theme.

2. Configure Developer Hub to load the theme in the UI by using the **themes** configuration field:

app-config.yaml fragment

```
dynamicPlugins:
  frontend:
    example.my-custom-theme-plugin:
      themes:
        - id: light (1)
          title: Light
          variant: light
          icon: someIconReference
          importName: lightThemeProvider
```

(1) Set your theme ID by specifying the desired value. Optionally, override the default Developer Hub themes by using the following ID values: **light** to replace the default light theme, or **dark** to replace the default dark theme.

Verification

- The theme is available in the Developer Hub **Settings** page.

9.12. CUSTOM COMPONENT OPTIONS FOR YOUR DEVELOPER HUB INSTANCE

There are two component variants that you can use to customize various components of your Developer Hub theme:

- **Patternfly**
- **MUI**

In addition to assigning a component variant to each parameter in the light or dark theme mode configurations, you can toggle the **rippleEffect on** or **off**.

The following code shows the options that you can use in the **app-config.yaml** file to configure the theme components for your Developer Hub instance:

```
app:
  branding:
    theme:
      light:
        options:
          rippleEffect: off / on
```

```
    paper: patternfly / mui
    buttons: patternfly / mui
    inputs: patternfly / mui
    accordions: patternfly / mui
    sidebars: patternfly / mui
    pages: patternfly / mui
    headers: patternfly / mui
    toolbars: patternfly / mui
    dialogs: patternfly / mui
    cards: patternfly / mui
    tables: patternfly / mui
    tabs: patternfly / mui
dark:
  options:
    rippleEffect: off / on
    paper: patternfly / mui
    buttons: patternfly / mui
    inputs: patternfly / mui
    accordions: patternfly / mui
    sidebars: patternfly / mui
    pages: patternfly / mui
    headers: patternfly / mui
    toolbars: patternfly / mui
    dialogs: patternfly / mui
    cards: patternfly / mui
    tables: patternfly / mui
    tabs: patternfly / mui
```

CHAPTER 10. CUSTOMIZING THE HOME PAGE

When using the **app-config**, you can do the following:

- Customize and extend the default Home page layout with additional cards that appear based on the plugins you have installed and enabled.
- Change quick access links.
- Add, reorganize, and remove the following available cards:
  - Search bar
  - Quick access
  - Headline
  - Markdown
  - Placeholder
  - Catalog starred entities
  - Featured docs

10.1. CUSTOMIZING THE HOME PAGE CARDS

Administrators can change the fixed height of cards that are in a 12-column grid.

The default Home page is as shown in the following **app-config.yaml** file configuration:

```
dynamicPlugins:
  frontend:
    red-hat-developer-hub.backstage-plugin-dynamic-home-page:
      dynamicRoutes:
        - path: /
          importName: DynamicHomePage
      mountPoints:
        - mountPoint: home.page/cards
          importName: SearchBar
          config:
            layouts:
              xl: { w: 10, h: 1, x: 1 }
              lg: { w: 10, h: 1, x: 1 }
              md: { w: 10, h: 1, x: 1 }
              sm: { w: 10, h: 1, x: 1 }
              xs: { w: 12, h: 1 }
              xxs: { w: 12, h: 1 }
        - mountPoint: home.page/cards
          importName: QuickAccessCard
          config:
            layouts:
              xl: { w: 7, h: 8 }
              lg: { w: 7, h: 8 }
              md: { w: 7, h: 8 }
              sm: { w: 12, h: 8 }
```

```
      xs: { w: 12, h: 8 }
      xxs: { w: 12, h: 8 }
  - mountPoint: home.page/cards
    importName: CatalogStarredEntitiesCard
    config:
      layouts:
        xl: { w: 5, h: 4, x: 7 }
        lg: { w: 5, h: 4, x: 7 }
        md: { w: 5, h: 4, x: 7 }
        sm: { w: 12, h: 4 }
        xs: { w: 12, h: 4 }
        xxs: { w: 12, h: 4 }
```

Prerequisites

- You have administrative access and can modify the **app-config.yaml** file for dynamic plugin configurations.

Procedure

- Configure different cards for your Home page in Red Hat Developer Hub as follows:

  #### Search

  ```
  dynamicPlugins:
    frontend:
      red-hat-developer-hub.backstage-plugin-dynamic-home-page:
        mountPoints:
          - mountPoint: home.page/cards
            importName: SearchBar
            config:
              layouts:
                xl: { w: 10, h: 1, x: 1 }
                lg: { w: 10, h: 1, x: 1 }
                md: { w: 10, h: 1, x: 1 }
                sm: { w: 10, h: 1, x: 1 }
                xs: { w: 12, h: 1 }
                xxs: { w: 12, h: 1 }
  ```

  Table 10.1. Available props

Prop	Default	Description
path	**/search**	Override the linked search path if needed
queryParam	**query**	Override the search query parameter name if needed

  #### Quick access

  ```
  dynamicPlugins:
    frontend:
  ```

```
    red-hat-developer-hub.backstage-plugin-dynamic-home-page:
      mountPoints:
        - mountPoint: home.page/cards
          importName: QuickAccessCard
          config:
            layouts:
              xl: { h: 8 }
              lg: { h: 8 }
              md: { h: 8 }
              sm: { h: 8 }
              xs: { h: 8 }
              xxs: { h: 8 }
```

Table 10.2. Available props

Prop	Default	Description
title	**Quick Access**	Override the linked search path if needed
path	none	Override the search query parameter name if needed

Headline

```
dynamicPlugins:
  frontend:
    red-hat-developer-hub.backstage-plugin-dynamic-home-page:
      mountPoints:
        - mountPoint: home.page/cards
          importName: Headline
          config:
            layouts:
              xl: { h: 1 }
              lg: { h: 1 }
              md: { h: 1 }
              sm: { h: 1 }
              xs: { h: 1 }
              xxs: { h: 1 }
            props:
              title: Important info
```

Table 10.3. Available props

Prop	Default	Description
title	none	Title

Markdown

```
dynamicPlugins:
```

```
  frontend:
    red-hat-developer-hub.backstage-plugin-dynamic-home-page:
      mountPoints:
        - mountPoint: home.page/cards
          importName: MarkdownCard
          config:
            layouts:
              xl: { w: 6, h: 4 }
              lg: { w: 6, h: 4 }
              md: { w: 6, h: 4 }
              sm: { w: 6, h: 4 }
              xs: { w: 6, h: 4 }
              xxs: { w: 6, h: 4 }
            props:
              title: Company links
              content: |
                ### RHDH
                * [Website](https://developers.redhat.com/rhdh/overview)
                * [Documentation]
(https://docs.redhat.com/en/documentation/red_hat_developer_hub/)
                * [Janus Plugins](https://github.com/janus-idp/backstage-plugins)
                * [Backstage Community Plugins](https://github.com/backstage/community-
plugins)
                * [RHDH Plugins](https://github.com/redhat-developer/rhdh-plugins)
                * [RHDH Showcase](https://github.com/redhat-developer/rhdh)
        - mountPoint: home.page/cards
          importName: Markdown
          config:
            layouts:
              xl: { w: 6, h: 4, x: 6 }
              lg: { w: 6, h: 4, x: 6 }
              md: { w: 6, h: 4, x: 6 }
              sm: { w: 6, h: 4, x: 6 }
              xs: { w: 6, h: 4, x: 6 }
              xxs: { w: 6, h: 4, x: 6 }
            props:
              title: Important company links
              content: |
                ### RHDH
                * [Website](https://developers.redhat.com/rhdh/overview)
                * [Documentation]
(https://docs.redhat.com/en/documentation/red_hat_developer_hub/)
                * [Documentation]
(https://docs.redhat.com/en/documentation/red_hat_developer_hub/)
                * [Janus Plugins](https://github.com/janus-idp/backstage-plugins)
                * [Backstage Community Plugins](https://github.com/backstage/community-
plugins)
                * [RHDH Plugins](https://github.com/redhat-developer/rhdh-plugins)
                * [RHDH Showcase](https://github.com/redhat-developer/rhdh)
```

Placeholder

```
dynamicPlugins:
  frontend:
    red-hat-developer-hub.backstage-plugin-dynamic-home-page:
      mountPoints:
```

```
      - mountPoint: home.page/cards
        importName: Placeholder
        config:
          layouts:
            xl: { w: 1, h: 1 }
            lg: { w: 1, h: 1 }
            md: { w: 1, h: 1 }
            sm: { w: 1, h: 1 }
            xs: { w: 1, h: 1 }
            xxs: { w: 1, h: 1 }
          props:
            showBorder: true
            debugContent: '1'
```

Catalog starred entities

```
dynamicPlugins:
  frontend:
    red-hat-developer-hub.backstage-plugin-dynamic-home-page:
      mountPoints:
        - mountPoint: home.page/cards
          importName: CatalogStarredEntitiesCard
```

Featured docs

```
dynamicPlugins:
  frontend:
    red-hat-developer-hub.backstage-plugin-dynamic-home-page:
      mountPoints:
        - mountPoint: home.page/cards
          importName: FeaturedDocsCard
```

10.2. DEFINING THE LAYOUT OF THE RED HAT DEVELOPER HUB HOME PAGE

The Home page uses a 12-column grid to position your cards. You can use the optimal parameters to define the layout of your Developer Hub Home page.

Prerequisites

- Include the following optimal parameters in each of your breakpoints:
  - width (w)
  - height (h)
  - position (x and y)

Procedure

- Configure your Developer Hub **app-config.yaml** configuration file by choosing one of the following options:

- Use the full space on smaller windows and half of the space on larger windows as follows:

```
dynamicPlugins:
  frontend:
    red-hat-developer-hub.backstage-plugin-dynamic-home-page:
      mountPoints:
        - mountPoint: home.page/cards
          importName: Placeholder
          config:
            layouts:
              xl: { w: 6, h: 2 }
              lg: { w: 6, h: 2 }
              md: { w: 6, h: 2 }
              sm: { w: 12, h: 2 }
              xs: { w: 12, h: 2 }
              xxs: { w: 12, h: 2 }
            props:
              showBorder: true
              debugContent: a placeholder
```

- Show the cards side by side by defining the **x** parameter as shown in the following example:

```
dynamicPlugins:
  frontend:
    red-hat-developer-hub.backstage-plugin-dynamic-home-page:
      mountPoints:
        - mountPoint: home.page/cards
          importName: Placeholder
          config:
            layouts:
              xl: { w: 6, h: 2 }
              lg: { w: 6, h: 2 }
              md: { w: 6, h: 2 }
              sm: { w: 12, h: 2 }
              xs: { w: 12, h: 2 }
              xxs: { w: 12, h: 2 }
            props:
              showBorder: true
              debugContent: left
        - mountPoint: home.page/cards
          importName: Placeholder
          config:
            layouts:
              xl: { w: 6, h: 2, x: 6 }
              lg: { w: 6, h: 2, x: 6 }
              md: { w: 6, h: 2, x: 6 }
              sm: { w: 12, h: 2, x: 0 }
              xs: { w: 12, h: 2, x: 0 }
              xxs: { w: 12, h: 2, x: 0 }
            props:
              showBorder: true
              debugContent: right
```

However, you can see a second card below this card by default.

- Show the cards in three columns by defining the **x** parameter as shown in the following example:

```
dynamicPlugins:
  frontend:
    red-hat-developer-hub.backstage-plugin-dynamic-home-page:
      mountPoints:
        - mountPoint: home.page/cards
          importName: Placeholder
          config:
            layouts:
              xl: { w: 4, h: 2 }
              lg: { w: 4, h: 2 }
              md: { w: 4, h: 2 }
              sm: { w: 6, h: 2 }
              xs: { w: 12, h: 2 }
              xxs: { w: 12, h: 2 }
            props:
              showBorder: true
              debugContent: left
        - mountPoint: home.page/cards
          importName: Placeholder
          config:
            layouts:
              xl: { w: 4, h: 2, x: 4 }
              lg: { w: 4, h: 2, x: 4 }
              md: { w: 4, h: 2, x: 4 }
              sm: { w: 6, h: 2, x: 6 }
              xs: { w: 12, h: 2 }
              xxs: { w: 12, h: 2 }
            props:
              showBorder: true
              debugContent: center
        - mountPoint: home.page/cards
          importName: Placeholder
          config:
            layouts:
              xl: { w: 4, h: 2, x: 8 }
              lg: { w: 4, h: 2, x: 8 }
              md: { w: 4, h: 2, x: 8 }
              sm: { w: 6, h: 2 }
              xs: { w: 12, h: 2 }
              xxs: { w: 12, h: 2 }
            props:
              showBorder: true
              debugContent: right
```

CHAPTER 11. CUSTOMIZING THE QUICK ACCESS CARD

To access the Home page in Red Hat Developer Hub, the base URL must include the **/developer-hub** proxy. You can configure the Home page by passing the data into the **app-config.yaml** file as a proxy. You can provide data to the Home page from the following sources:

- JSON files hosted on GitHub or GitLab.
- A dedicated service that provides the Home page data in JSON format using an API.

11.1. USING HOSTED JSON FILES TO PROVIDE DATA TO THE QUICK ACCESS CARD

Prerequisites

- You have installed Red Hat Developer Hub by using either the Operator or Helm chart. See Installing Red Hat Developer Hub on OpenShift Container Platform .

Procedure

- To access the data from the JSON files, add the following code to the **app-config.yaml** Developer Hub configuration file:
- Add the following code to 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 /redhat-developer/rhdh/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
```

11.2. USING A DEDICATED SERVICE TO PROVIDE DATA TO THE QUICK ACCESS CARD

When using a dedicated service, you can do the following tasks:

- Use the same service to provide the data to all configurable Developer Hub pages or use a different service for each page.
- Use the **red-hat-developer-hub-customization-provider** as an example service, which provides data for both the Home and Tech Radar pages. 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.

- Deploy the **red-hat-developer-hub-customization-provider** service and the Developer Hub Helm chart on the same cluster.

Prerequisites

- You have installed the Red Hat Developer Hub using Helm chart. For more information, see Installing Red Hat Developer Hub on OpenShift Container Platform with the Helm chart .

Procedure

To use a separate service to provide the Home page data, complete the following steps:

1. From the **Developer** perspective in the Red Hat OpenShift Container Platform web console, click **+Add** > **Import from Git**.
2. Enter the URL of your Git repository into the **Git Repo URL** field.
   To use the **red-hat-developer-hub-customization-provider** service, add the URL for the red-hat-developer-hub-customization-provider repository or your fork of the repository containing your customizations.
3. On the **General** tab, enter **red-hat-developer-hub-customization-provider** in the **Name** field and click **Create**.
4. On the **Advanced Options** tab, copy the value from **Target Port**.

NOTE

Target Port automatically generates a Kubernetes or OpenShift Container Platform service to communicate with.

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

```
proxy:
  endpoints:
    # Other Proxies
    # customize developer hub instance
    '/developer-hub':
      target: ${HOMEPAGE_DATA_URL} (1)
      changeOrigin: true
      # Change to "false" in case of using self-hosted cluster with a self-signed certificate
      secure: true
```

(1) **http://<SERVICE_NAME>:8080**, for example, **http://rhdh-customization-provider:8080**.

NOTE

The **red-hat-developer-hub-customization-provider** service contains the 8080 port by default. If you are using a custom port, you can specify it with the 'PORT' environmental variable in the **app-config.yaml** file.

6. Replace the **HOMEPAGE_DATA_URL** by adding the URL to **rhdh-secrets** or by directly replacing it in your custom ConfigMap.

7. Delete the Developer Hub pod to ensure that the new configurations are loaded correctly.

Verification

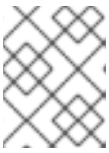
- To view the service, go to the **Administrator** perspective in the OpenShift Container Platform web console and click **Networking** > **Service**.

NOTE

You can also view **Service Resources** in the Topology view.

- Ensure that the provided API URL for the Home page returns 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"
      }
    ]
  }
]
```



NOTE

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

- If the images or icons do not load, then allowlist them by adding your image or icon host URLs to the content security policy (csp) **img-src** in your custom ConfigMap as shown in the following example:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app-config.yaml
data:
  app-config.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
```