Red Hat Developer Hub 1.1

Red Hat Developer Hub 入门

Last Updated: 2024-06-26

法律通告

摘要

本文档介绍了安装和配置 Red Hat Developer Hub 的要求和说明。

目录

前言

作为开发者，您可以使用 Red Hat Developer Hub 体验简化的开发环境。Red Hat Developer Hub 由集中软件目录驱动，为您的微服务和基础架构提供效率。它使您的产品团队能够在没有任何影响的情况下提供质量代码。

RED HAT DEVELOPER HUB 支持

如果您在执行本文档所述的某个流程时遇到问题，请访问红帽客户门户。您可以使用红帽客户门户网站进行以下目的：

- 搜索或浏览红帽知识库，了解有关红帽产品的技术支持文章。
- 为红帽全球支持服务(GSS)创建支持问题单。https://access.redhat.com/support/cases/#/case/new/get-support?caseCreate=true要创建支持问题单，请选择 **Red Hat Developer Hub**作为产品，然后选择适当的产品版本。

第 1 章 RED HAT DEVELOPER HUB 概述

Red Hat Developer Hub (Developer Hub)充当专为构建开发人员门户设计的开放开发人员平台。通过开发人员 Hub，工程团队可以访问统一平台，以简化开发流程，并提供各种工具和资源，以便有效地构建高质量的软件。

Developer Hub 的目标是通过以下方式解决与创建和持续开发人员门户相关的困难：

- 中央化仪表板，用于查看所有可用的开发人员工具和资源，以提高生产力
- 自助服务功能以及 guardrails，适用于符合企业级最佳实践的云原生应用程序开发
- 对整个企业的所有开发人员进行适当的安全性和监管

Red Hat Developer Hub 通过提供开发人员体验来简化对内部批准的工具、编程语言以及自助管理门户网站中各种开发人员资源的决策。这种方法有助于加速应用程序开发和维护代码质量，同时促进创新。

第 2 章 安装 RED HAT DEVELOPER HUB

管理用户可以配置角色、权限和其他设置，以便其他授权用户可以在多个平台上安装 Red Hat Developer Hub。您可以使用 Helm chart 或 Red Hat Developer Hub Operator 安装 Developer Hub。

有关安装 Developer Hub 的更多信息，请参阅 Red Hat Developer Hub 管理指南 。

第 3 章 RED HAT DEVELOPER HUB 支持的配置

本节论述了访问 Red Hat Developer Hub 所需的配置，包括：

- 自定义应用程序配置
- Developer Hub 目录的源控制配置

3.1. 在 RED HAT OPENSHIFT CONTAINER PLATFORM 中添加自定义应用程序配置文件

要访问 Red Hat Developer Hub，您必须将自定义应用程序配置文件添加到 OpenShift 中。在 OpenShift Container Platform 中，您可以使用以下内容作为基础模板，创建名为 **app-config-rhdh** 的 ConfigMap：

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

先决条件

- 您已创建了 Red Hat OpenShift Container Platform 帐户。

流程

1. 在 OpenShift Container Platform Web 控制台中，选择 **ConfigMaps** 选项卡。
2. 点 **Create ConfigMap**。
3. 在 **Create ConfigMap** 页面中，选择 **Configure via** 中的 **YAML view** 选项，并根据需要对文件进行更改。
4. 点 **Create**。
5. 进入 **Helm** 标签页。
   Helm Releases 列表会出现在页面中。
6. 点 Helm 发行版本的 overflow 菜单并选择 **Upgrade**。
7. 使用以下视图之一编辑 Helm 配置：
   - 使用 **Form view**
     a. 展开 **Root Schema → Backstage chart schema → Backstage parameters → Extra app 配置文件到内联到命令参数 中**。
     b. 点 **Add Extra app 配置文件以内联到命令参数**链接。
     c. 在以下字段中输入值：

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

d. 单击 **Upgrade**。

- 使用 YAML 视图

a. 使用以下方法设置 **upstream.backstage.extraAppConfig.configMapRef** 和 **upstream.backstage.extraAppConfig.filename** 参数的值：

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

b. 单击 **Upgrade**。

3.2. 在 RED HAT DEVELOPER HUB 中为目录添加源控制

要在 Red Hat Developer Hub 中填充目录，您需要添加软件模板，并添加模板，您必须启用源控制，如 GitHub、GitLab 或 BitBucket。

先决条件

- 您有一个 GitHub 帐户。
- 在 Red Hat OpenShift 集群上有一个帐户。
- 已安装 Developer Hub，否则 GitHub 登录失败。有关安装的详情，请参考 *第 2 章 安装 Red Hat Developer Hub*。

3.2.1. 配置 GitHub 身份验证

需要 GitHub 身份验证配置，才能在 Developer Hub 中启用 GitHub OAuth 登录。

流程

1. 在 Red Hat OpenShift 集群中，导航到您要创建 OAuth 应用程序的 GitHub 机构的主页面。
2. 点 **Settings → Developer Settings → OAuth Apps → Register an application**
3. 输入应用程序名称作为 **Developer Hub**。
4. 将以下 URL 添加为 **Homepage URL** ：
   https://developer-hub-<NAMESPACE_NAME>.<OPENSHIFT_ROUTE_HOST>/
5. 将以下 URL 添加为 **授权回调 URL** ：
   https://developer-hub-<NAMESPACE_NAME>.<OPENSHIFT_ROUTE_HOST>/api/auth/github/handler/frame
6. 清除 **Enable Device Flow**复选框。

7. 点 **Register application** 创建 OAuth 应用程序。
8. 创建应用程序后，点 **Generate a new client secret** 并复制生成的客户端 secret。
9. 在 OpenShift 中，单击 **ConfigMaps**。
10. 使用提供的环境变量作为键，生成名为 'github-secrets' 的键/值 secret，然后输入您为 GitHub OAuth 应用程序生成的值：
    a. 在 Red Hat OpenShift 中，进入 **Secrets** 选项卡，再点 **Create**。
    b. 选择 **Key/value secret**。
    c. 输入 **Secret 名称作为 github-secrets**。
    d. 将环境变量添加为 **Key** 和 **Value**，再单击 **Create**。

    表 3.1. 环境变量

键	值
GITHUB_OAUTH_CLIENT_ID	来自 OAuth 应用程序的客户端 ID
GITHUB_OAUTH_CLIENT_SECRET	OAuth 应用程序的客户端 Secret

11. 修改 **app-config-rhdh** ConfigMap 使其包含 GitHub 身份验证配置，如下所示：

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

12. 点击 **Save**。
13. 进入 **Helm** 选项卡并选择 **Upgrade**。
14. 使用以下视图之一编辑 Helm 配置：
    - 使用 **Form view**
      a. 展开 **Root Schema → Backstage Chart Schema → Backstage Parameters → Backstage container 环境变量**。

b. **从现有的 Secrets 链接点 Add Backstage容器环境变量。**

c. 输入 **github-secrets** 作为值。

d. 单击 **Upgrade**。

- 使用 YAML 视图

a. 将 **upstream.backstage.extraEnvVarsSecrets** 的值设置为 **github-secrets**，如下例所示：

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

b. 单击 **Upgrade**。

3.2.2. 配置 GitHub 集成

在 Developer Hub 中启用 GitHub 插件需要配置 GitHub。

流程

1. 在 Red Hat OpenShift 集群中，导航到您要创建 OAuth 应用程序的 GitHub 机构的主页面。
2. 点 **Settings → Developer Settings → GitHub Apps → New GitHub App**。
3. 输入应用程序名称作为 **Developer Hub**。
4. 将以下 URL 添加为 **Homepage URL** ：
   https://developer-hub-<NAMESPACE_NAME>.<OPENSHIFT_ROUTE_HOST>/
5. 将以下 URL 添加为 **授权回调 URL** ：
   https://developer-hub-<NAMESPACE_NAME>.
   <OPENSHIFT_ROUTE_HOST>/api/auth/github/handler/frame
6. 取消选择 **Webhook URL → Active**。
7. 在 **可以安装此 GitHub 应用程序的位置下，请确保只选择此帐户**。
8. 点击 **Register application**。
9. 创建应用程序后，点 **Generate a new client secret** 并复制生成的客户端 secret。
10. 点页面底部的 **Generate a私钥** 并下载生成的文件。
11. 在 OpenShift 中，单击 **ConfigMaps**。
12. 使用提供的环境变量作为键，生成名为 'github-secrets' 的键/值 secret，然后输入您为 GitHub OAuth 应用程序生成的值：

    a. 在 Red Hat OpenShift 中，进入 **Secrets** 选项卡，再点 **Create**。

b. 选择 **Key/value secret**。

c. 输入 **Secret 名称作为 github-secrets**。

d. 将环境变量添加为 **Key** 和 **Value**，再单击 **Create**。

表 3.2. 环境变量

键	值
GITHUB_APP_APP_ID	来自 GitHub 应用程序的应用程序 ID
GITHUB_APP_CLIENT_ID	来自 GitHub 应用程序的客户端 ID
GITHUB_APP_CLIENT_SECRET	GitHub 应用程序的客户端 Secret
GITHUB_APP_WEBHOOK_URL	输入 "none"
GITHUB_APP_WEBHOOK_SECRET	输入 "none"
GITHUB_APP_PRIVATE_KEY	上传下载的私钥

13. 修改 **app-config-rhdh** ConfigMap 使其包含 GitHub 集成配置，如下所示：

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

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

3.2.3. 在 Red Hat Developer Hub 中启用 GitHub 发现

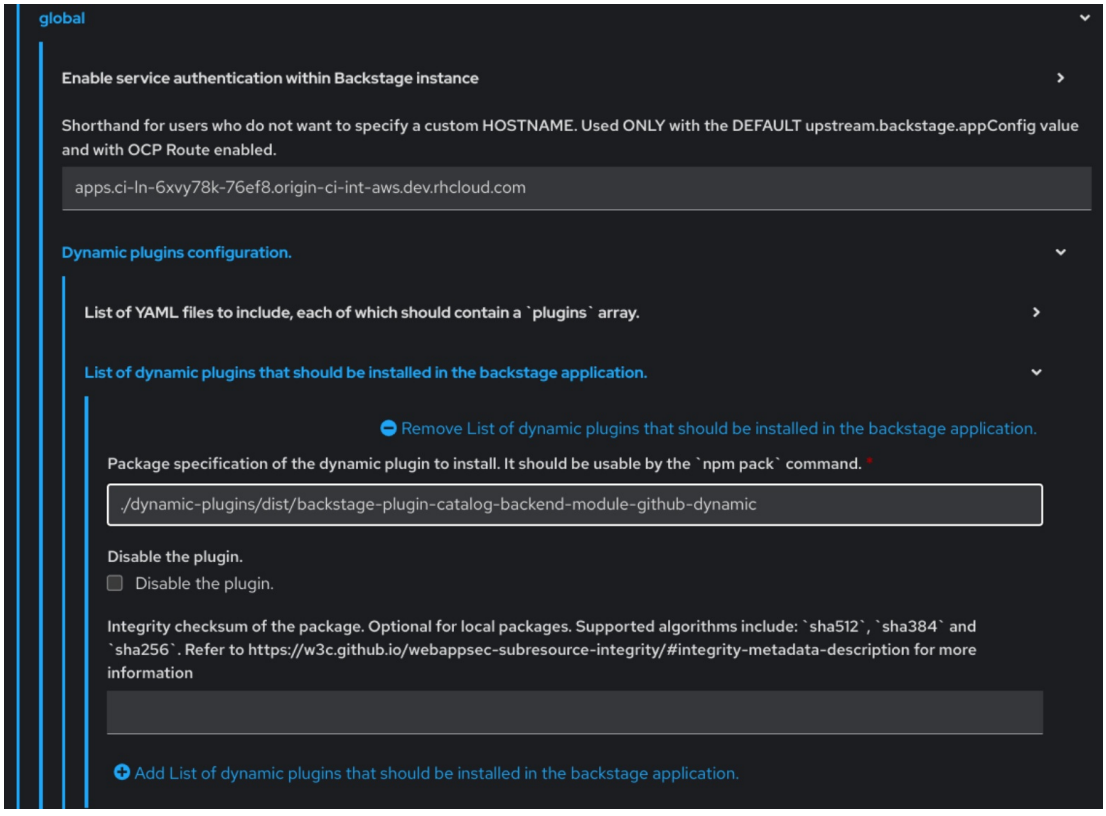
您可以为 Developer Hub 中的组件启用 GitHub 发现功能，如包含 **catalog-info.yaml** 文件的任何存储库。

先决条件

- 您已设置 GitHub 集成。如需更多信息，请参阅 第 3.2.2 节 “配置 GitHub 集成”。

流程

1. 在 OpenShift Container Platform Web 控制台的 **Developer** 视角中，进入 **Helm** 选项卡。
2. 点 Helm 发行版本的 overflow 菜单并选择 **Upgrade**。
3. 使用以下视图之一编辑 Helm 配置：
   - 使用 **Form view**
     a. 展开 **Root Schema → global → Dynamic plugins configuration → 在 backstage 应用中安装的动态插件列表**。
     b. 单击 **backstage 应用链接中应安装的动态插件的 Add List**。
     c. 在要安装的动态插件的 **Package 规格中。它应该可以被 npm pack 命令使用。** 字段，添加以下值：
     ./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-dynamic



     d. 单击 **Upgrade**。
   - 使用 **YAML 视图**
     a. 将 **global.dynamic.plugins.package** 参数的值设置为 **./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-dynamic**，如下例所示：

```
global:
  dynamic:
    # other Red Hat Developer Hub Helm Chart configurations
    plugins:
```

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

b. 单击 **Upgrade**。

4. 在 ConfigMap 中添加以下代码：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app-config-rhdh
data:
  app-config-rhdh.yaml: |
    ...
      catalog:
        providers:
          github:
            providerId:
              organization: '${GITHUB_ORG}'
              schedule:
                frequency:
                  minutes: 30
                initialDelay:
                  seconds: 15
                timeout:
                  minutes: 3
    ...
```

在前面的代码中，将 **${GITHUB_ORG}** 替换为您要发现组件的 GitHub 组织。另外，如果只有一个供应商，可以在 ConfigMap 中添加以下代码：

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

当存在列表时，需要前面的代码中的 **providerId** 来识别提供程序。

5. 点击 **Save**。

3.2.4. 在 Red Hat Developer Hub 中启用 GitHub 机构成员发现

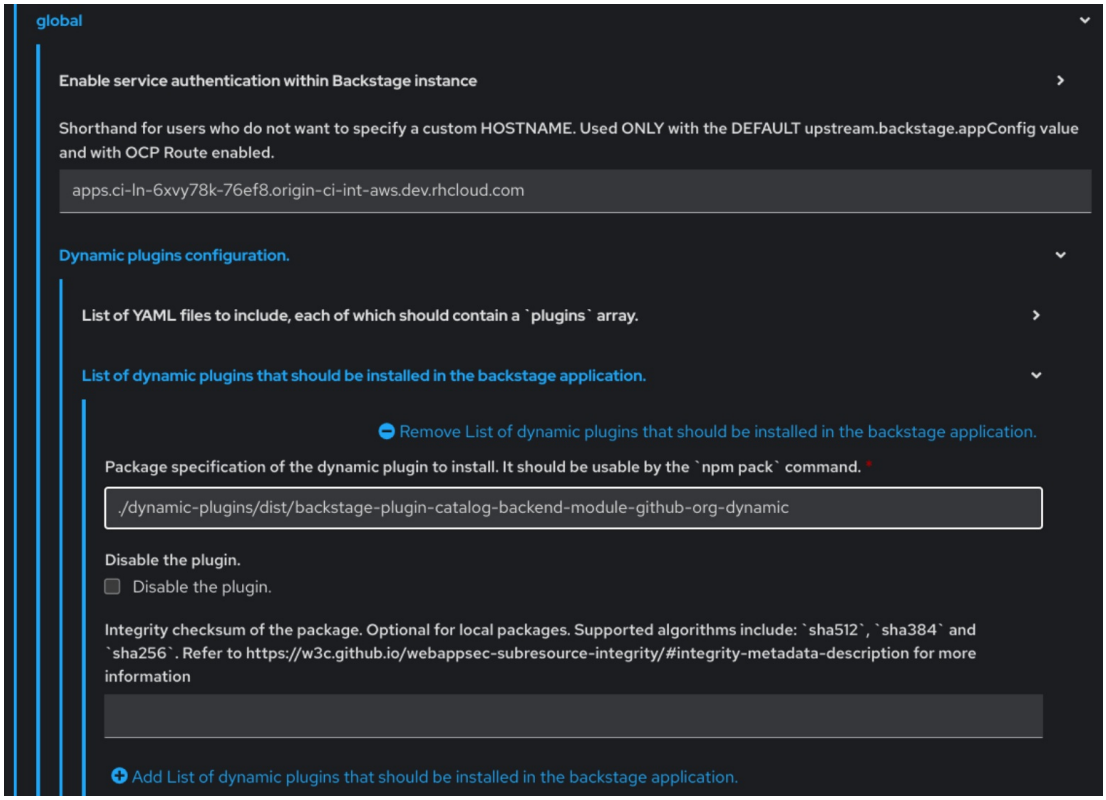
您还可以为 GitHub 组织的成员启用 GitHub 发现功能。

先决条件

- 您已设置 GitHub 集成。如需更多信息，请参阅 第 3.2.2 节 "配置 GitHub 集成"。

流程

1. 在 OpenShift Container Platform Web 控制台的 **Developer** 视角中，进入 **Helm** 选项卡。
2. 点 Helm 发行版本的 overflow 菜单并选择 **Upgrade**。
3. 使用以下视图之一编辑 Helm 配置：
   - 使用 **Form view**
     a. 展开 **Root Schema → global → Dynamic plugins configuration → 在 backstage 应用中安装的动态插件列表**。
     b. 单击 **backstage 应用链接中应安装的动态插件的 Add List**。
     c. 在要安装的动态插件的 **Package 规格中。它应该可以被 npm pack 命令使用。** 字段，添加以下值：
     ./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-org-dynamic



     d. 单击 **Upgrade**。
   - 使用 **YAML 视图**

a. 将 **global.dynamic.plugins.package** 参数的值设置为 **./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-org-dynamic**，如下例所示：

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

b. 单击 **Upgrade**。

4. 在 ConfigMap 中添加以下代码：

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

其中：

${GITHUB_URL}

表示必须替换为 GitHub URL 的变量。

${GITHUB_ORG}

表示必须替换为您要从中嵌套用户的 GitHub 组织的变量。

5. 点击 **Save**。

第 4 章 自定义 RED HAT DEVELOPER HUB 中的主页

在 Red Hat Developer Hub 中，Home 页面数据是可配置的，它可以作为代理传递给 **app-config.yaml** 文件。您可以使用以下方法提供主页数据：

- 使用托管或 GitHub 或 GitLab 的 JSON 文件。要从 JSON 文件访问数据，您可以在 **app-config.yaml** 文件中添加以下代码：

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

- 使用单独的服务，使用 API 以 JSON 格式提供 Home 页面数据。

注意

不需要同一服务提供 Home 页面和 Tech Radar 数据。

您可以使用 **red-hat-developer-hub-customization-provider** 作为示例服务，该服务为 Home page 和 Tech Radar 提供数据。**red-hat-developer-hub-customization-provider** 服务提供与默认的 Developer Hub 数据相同的数据。如果需要，您可以从 GitHub 中 fork **red-hat-developer-hub-customization-provider** 服务存储库，并使用您自己的数据进行修改。

本节论述了如何将 **red-hat-developer-hub-customization-provider** 服务部署到部署 Developer Hub Helm Chart 的集群。

先决条件

- 已使用 Helm Chart 安装 Red Hat Developer Hub。如需更多信息，请参阅 *第 2 章 安装 Red Hat Developer Hub*。

流程

1. 在 Red Hat OpenShift 中，选择 **+Add** 并点 **Import from Git** 选项。
2. 将 Git 存储库的 URL 添加到 **Git Repo URL** 字段中。
   要使用 **red-hat-developer-hub-customization-provider** 服务，您可以添加 red-hat-developer-hub-customization-provider 存储库的 URL。
3. 在 **General** 部分中，将 **Name** 字段中的值重命名为 **rhdh-customization-provider**，再单击 **Create**。

4. 前往 **Advanced Options**，再从 **Target Port** 复制值。
Target Port 用于自动生成 Kubernetes 或 OpenShift 服务以与之进行通信。

5. 要查看服务，请导航到 **OpenShift Administrator** 视图，再前往 **Networking → Service** 部分。
您还可以在 Topology 视图中查看 **Service Resources**。

   如果您使用示例遵循这个步骤，则调用 **rhdh-customization-provider** 服务，并包含 8080 端口。为 Home 页面提供的 API URL 必须以 JSON 格式返回数据，如下例所示：

```
[
  {
    "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"
      }
    ]
  }
]
```

   如果请求调用失败或没有配置，Developer Hub 实例会回退到默认的本地数据。

   要访问 Red Hat Developer Hub 中的 Home 页面，基本 URL 必须包含 **/developer-hub** 代理。

6. 将以下代码添加到 **app-config-rhdh.yaml** 文件中：

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

   确保 API 请求调用以 JSON 格式返回响应。

7. 将 **HOMEPAGE_DATA_URL** 定义为 **http://<SERVICE_NAME>:8080。例如，'http://rhdh-customization-provider:8080**。
   您可以通过将 URL 添加到 **rhdh-secrets** 或直接在自定义 ConfigMap 中替换它来替换 **HOMEPAGE_DATA_URL**。

8. 删除 Developer Hub Pod 以拉取更改。
   如果镜像或图标没有加载，则通过将镜像或图标主机 URL 添加到自定义 ConfigMap 中的内容安全策略(csp) **img-src** 中列入白名单，如下所示：

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

   之后，删除 pod 以确保正确载入新配置。

第 5 章 在 RED HAT DEVELOPER HUB 中自定义 TECH RADAR 页面

在 Red Hat Developer Hub 中，不能使用 Helm Chart 中的动态插件功能启用 Tech Radar 页面。

与 Home page 自定义类似，基础技术 Radar URL 必须包含 **/developer-hub/tech-radar** 代理。您可以使用以下方法提供 Tech Radar 页面数据：

- 使用托管或 GitHub 或 GitLab 的 JSON 文件。要从 JSON 文件访问数据，您可以在 **app-config.yaml** 文件中添加以下代码：

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

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

注意

随着在用于 **tech-radar** 和 **homepage** 快速访问代理的 **pathRewrites** 之间重叠，则 **tech-radar** (**^api/proxy/developer-hub/tech-radar**)的配置必须在 **主页** 配置之前存在。

有关在 Red Hat Developer Hub 中自定义 Home 页面的更多信息，请参阅 第 4 章 *自定义 Red Hat Developer Hub 中的主页*。

- **使用单独的服务，使用 API 以 JSON 格式提供 Tech Radar 数据。**

先决条件

- **已使用 Helm Chart 安装 Red Hat Developer Hub。如需更多信息，请参阅 第 2 章 *安装 Red Hat Developer Hub*。**

流程

1. 将以下代码添加到 app-config-rhdh.yaml 文件中：

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

确保 API 请求调用以 JSON 格式返回响应。

2. 将 TECHRADAR_DATA_URL' 定义为 http://<SERVICE_NAME>/tech-radar，例如 http://rhdh-customization-provider/tech-radar。

注意

您可以通过将其添加到 rhdh-secrets 或直接将其替换为自定义 ConfigMap 中的值来定义 TECHRADAR_DATA_URL。

3. 删除 Developer Hub Pod 以拉取更改。

第 6 章 RED HAT DEVELOPER HUB 中的其他自定义

本节论述了您可以应用到 Red Hat Developer Hub 的额外自定义选项。

自定义标签页提示

要自定义标签页提示，请在 app-config-rhdh.yaml 文件中添加以下内容：

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

自定义 Developer Hub 实例的品牌

要自定义 Developer Hub 实例品牌，请在 app-config-rhdh.yaml 文件中添加以下内容：

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

在前面的配置中，

- fullLogo 是展开(pinned)侧边栏上的徽标，并需要一个 base64 编码镜像。
- iconLogo 是折叠(unpinned)侧边栏上的徽标，并需要一个 base64 编码镜像。
- primaryColor 是链接的颜色，以及输入颜色的大部分按钮。primaryColor 支持的格式包括：
  - #nnn

- #nnnnnn
- rgb()
- rgba()
- hsl()
- hsla()
- color()

- headerColor1 （横幅旁边）和 headerColor2 （横幅的右边）会更改每个页面标题横幅的颜色，以及模板卡的横幅。headerColor1 和 headerColor2 支持的格式包括：
  - #nnn
  - #nnnnnn
  - rgb()
  - rgba()
  - hsl()
  - hsla()
  - color()

- navigationIndicatorColor 在侧边栏中更改指示您要所处的标签页的颜色。navigationIndicatorColor 支持的格式包括：
  - #nnn
  - #nnnnnn
  - rgb()
  - rgba()
  - hsl()
  - hsla()
  - color()

第 7 章 在 RED HAT DEVELOPER HUB 中自定义主题

您可以自定义 Red Hat Developer Hub (Developer Hub)主题模式。

RHDH 支持以下主题模式：

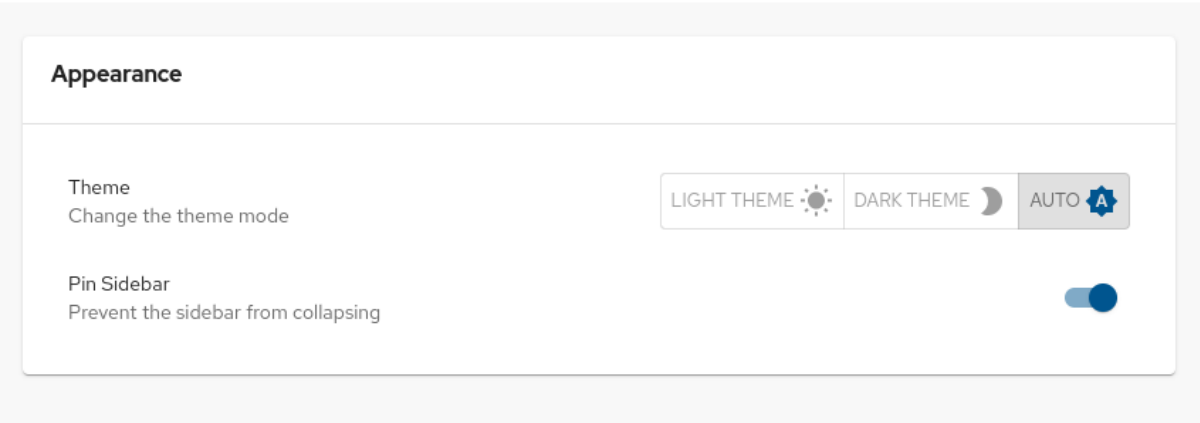
- 轻型主题（默认）
- dark 主题
- auto

先决条件

- 已登陆到 RHDH web 控制台。

流程

1. 单击 **Settings**。
2. 在 **Appearance** 面板中，单击 **LIGHT THEME**、**DARK THEME** 或 **AUTO** 以更改主题模式。



第 8 章 RED HAT DEVELOPER HUB 中的 SERVICENOW CUSTOM 操作

重要

这些功能仅用于技术预览。红帽产品服务级别协议（SLA）不支持技术预览功能，且其功能可能并不完善，因此红帽不建议在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能的更多信息，请参阅 技术预览功能范围。

在 Red Hat Developer Hub 中，您可以访问 ServiceNow 自定义操作（自定义操作），以便在目录中获取和注册资源。

Developer Hub 中的自定义操作可让您促进和自动化记录管理。使用自定义操作，您可以执行以下操作：

- 创建、更新或删除记录
- 检索有关单个记录或多个记录的信息

8.1. 在 RED HAT DEVELOPER HUB 中启用 SERVICENOW 自定义操作插件

在 Red Hat Developer Hub 中，ServiceNow 自定义操作作为预加载插件提供，该插件默认为禁用。您可以按照以下流程启用自定义操作插件。

先决条件

- Red Hat Developer Hub 已安装并运行。有关安装 Developer Hub 的更多信息，请参阅 *第 2 章 安装 Red Hat Developer Hub*。
- 您已在 Developer Hub 中创建项目。

流程

1. 要激活自定义操作插件，请使用插件名称 添加软件包，并更新 Helm Chart 中的 disabled 字

段，如下所示：

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

注意

插件的默认配置从 **dynamic-plugins.default.yaml** 文件中提取，但您可以使用 **pluginConfig** 条目来覆盖默认配置。

2. 在 **Helm Chart** 中设置以下变量来访问自定义操作：

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

8.2. RED HAT DEVELOPER HUB 中支持的 SERVICENOW 自定义操作

ServiceNow 自定义操作可让您管理 **Red Hat Developer Hub** 中的记录。自定义操作支持以下 **API** 请求的 **HTTP** 方法：

- **GET:** 从指定资源端点检索指定信息
- **POST** ：创建或更新资源
- **PUT** ：修改资源
- **PATCH** ：更新资源

- **DELETE ：删除资源**

8.2.1. ServiceNow 自定义操作

[GET] servicenow:now:table:retrieveRecord

从 Developer Hub 中的表检索指定记录的信息。

表 8.1. 输入参数

Name	类型	要求	描述
tableName	**string**	必填	要从中检索记录的表名称
sysId	**string**	必填	要检索的记录的唯一标识符
sysparmDisplayValue	**enum("true", "false", "all")**	选填	返回字段显示值，如 **true**，实际值为 **false**，或两者。默认值为 **false**。
sysparmExcludeReferenceLink	**布尔值**	选填	设置为 **true** 以排除参考字段的 Table API 链接。默认值为 **false**。
sysparmFields	**string[]**	选填	响应中返回的字段数组
sysparmView	**string**	选填	根据指定的 UI 视图呈现响应。您可以使用 **sysparm_fields** 覆盖此参数。
sysparmQueryNoDomain	**布尔值**	选填	如果授权，则设置为 **true** 以访问跨域的数据。默认值为 **false**。

表 8.2. 输出参数

Name	类型	描述
result	**record<PropertyKey, unknown>**	请求的响应正文

[GET] servicenow:now:table:retrieveRecords

从 Developer Hub 中的表检索有关多个记录的信息。

表 8.3. 输入参数

Name	类型	要求	描述
tableName	**string**	必填	要从中检索记录的表名称
sysparamQuery	**string**	选填	用于过滤结果的编码查询字符串
sysparmDisplayValue	**enum("true", "false", "all")**	选填	返回字段显示值，如 **true**，实际值为 **false**，或两者。默认值为 **false**。
sysparmExcludeReferenceLink	**布尔值**	选填	设置为 **true** 以排除参考字段的 Table API 链接。默认值为 **false**。
sysparmSuppressPaginationHeader	**布尔值**	选填	设置为 **true** 以阻止分页标头。默认值为 **false**。
sysparmFields	**string[]**	选填	响应中返回的字段数组
sysparmLimit	**int**	选填	每页返回的最大结果数。默认值为 **10,000**。
sysparmView	**string**	选填	根据指定的 UI 视图呈现响应。您可以使用 **sysparm_fields** 覆盖此参数。
sysparmQueryCategory	**string**	选填	用于查询的查询类别的名称
sysparmQueryNoDomain	**布尔值**	选填	如果授权，则设置为 **true** 以访问跨域的数据。默认值为 **false**。
sysparmNoCount	**布尔值**	选填	不要在表上执行 select count DomainMapping。默认值为 **false**。

表 8.4. 输出参数

Name	类型	描述
result	**record<PropertyKey, unknown>**	请求的响应正文

[POST] servicenow:now:table:createRecord

在 Developer Hub 的表中创建记录。

表 8.5. 输入参数

Name	类型	要求	描述
tableName	**string**	必填	要保存记录的表的名称
requestBody	**record<PropertyKey, unknown>**	选填	在指定记录中定义的每个参数的字段名称和关联的值
sysparmDisplayValue	**enum("true", "false", "all")**	选填	返回字段显示值，如 **true**，实际值为 **false**，或两者。默认值为 **false**。
sysparmExcludeReferenceLink	**布尔值**	选填	设置为 **true** 以排除参考字段的 Table API 链接。默认值为 **false**。
sysparmFields	**string[]**	选填	响应中返回的字段数组
sysparmInputDisplayValue	**布尔值**	选填	使用其显示值（如 **true** 或实际值）设置字段值，如 **false**。默认值为 **false**。
sysparmSuppressAutoSysField	**布尔值**	选填	设置为 **true**，以禁止自动生成系统字段。默认值为 **false**。
sysparmView	**string**	选填	根据指定的 UI 视图呈现响应。您可以使用 **sysparm_fields** 覆盖此参数。

表 8.6. 输出参数

Name	类型	描述
result	**record<PropertyKey, unknown>**	请求的响应正文

[PUT] servicenow:now:table:modifyRecord

修改 Developer Hub 中表中的记录。

表 8.7. 输入参数

Name	类型	要求	描述
tableName	**string**	必填	要从中修改记录的表名称
sysId	**string**	必填	要修改的记录的唯一标识符
requestBody	**record<PropertyKey, unknown>**	选填	在指定记录中定义的每个参数的字段名称和关联的值
sysparmDisplayValue	**enum("true", "false", "all")**	选填	返回字段显示值，如 **true**，实际值为 **false**，或两者。默认值为 **false**。
sysparmExcludeReferenceLink	**布尔值**	选填	设置为 **true** 以排除参考字段的 Table API 链接。默认值为 **false**。
sysparmFields	**string[]**	选填	响应中返回的字段数组
sysparmInputDisplayValue	**布尔值**	选填	使用其显示值（如 **true** 或实际值）设置字段值，如 **false**。默认值为 **false**。
sysparmSuppressAutoSysField	**布尔值**	选填	设置为 **true**，以禁止自动生成系统字段。默认值为 **false**。
sysparmView	**string**	选填	根据指定的 UI 视图呈现响应。您可以使用 **sysparm_fields** 覆盖此参数。
sysparmQueryNoDomain	**布尔值**	选填	如果授权，则设置为 **true** 以访问跨域的数据。默认值为 **false**。

表 8.8. 输出参数

Name	类型	描述
result	**record<PropertyKey, unknown>**	请求的响应正文

[PATCH] 服务现在：now:table:updateRecord

更新 Developer Hub 中的表中的记录。

表 8.9. 输入参数

Name	类型	要求	描述
tableName	**string**	必填	要更新记录的表的名称
sysId	**string**	必填	要更新的记录的唯一标识符
requestBody	**record<PropertyKey, unknown>**	选填	在指定记录中定义的每个参数的字段名称和关联的值
sysparmDisplayValue	**enum("true", "false", "all")**	选填	返回字段显示值，如 **true**，实际值为 **false**，或两者。默认值为 **false**。
sysparmExcludeReferenceLink	**布尔值**	选填	设置为 **true** 以排除参考字段的 Table API 链接。默认值为 **false**。
sysparmFields	**string[]**	选填	响应中返回的字段数组
sysparmInputDisplayValue	**布尔值**	选填	使用其显示值（如 **true** 或实际值）设置字段值，如 **false**。默认值为 **false**。
sysparmSuppressAutoSysField	**布尔值**	选填	设置为 **true**，以禁止自动生成系统字段。默认值为 **false**。
sysparmView	**string**	选填	根据指定的 UI 视图呈现响应。您可以使用 **sysparm_fields** 覆盖此参数。
sysparmQueryNoDomain	**布尔值**	选填	如果授权，则设置为 **true** 以访问跨域的数据。默认值为 **false**。

表 8.10. 输出参数

Name	类型	描述
result	**record<PropertyKey, unknown>**	请求的响应正文

[DELETE] servicenow:now:table:deleteRecord

从 Developer Hub 中的表中删除记录。

表 8.11. 输入参数

Name	类型	要求	描述
tableName	**string**	必填	从中删除记录的表名称
sysId	**string**	必填	要删除的记录的唯一标识符
sysparmQueryNoDomain	**布尔值**	选填	如果授权，则设置为 **true** 以访问跨域的数据。默认值为 **false**。

第 9 章 GITHUB 身份验证供应商

Red Hat Developer Hub 使用内置的 GitHub 身份验证供应商来验证 GitHub 或 GitHub Enterprise 中的用户。

9.1. GITHUB APP 概述

GitHub Apps 通常优先于 OAuth 应用程序，因为它们使用精细的权限，为应用程序提供可访问的存储库，并使用简短的令牌。如需更多信息，请参阅 GitHub 文档中的 GitHub 应用程序概述。

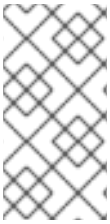
9.2. 注册 GITHUB 应用程序

在 GitHub App 中，您可以将允许的范围配置为该应用程序的一部分，因此您必须验证插件所需的范围。范围信息在插件 README 文件中提供。

要添加 GitHub 身份验证，请完成 GitHub 网站上 注册 GitHub App 中的步骤。

使用以下示例将有关您的生产环境的信息输入到 Register new GitHub App 页面中的必填字段中：

- 应用程序名称：Red Hat Developer Hub
- 主页 URL: https://developer-hub-<NAMESPACE_NAME>.<KUBERNETES_ROUTE_HOST>
- 授权回调 URL ：https://developer-hub-<NAMESPACE_NAME>.<KUBERNETES_ROUTE_HOST>/api/auth/github/handler/frame



注意

Homepage URL 指向 Developer Hub 前端，而授权回调 URL 指向身份验证供应商后端。

9.3. 在 DEVELOPER HUB 中配置 GITHUB 应用程序

要为 Developer Hub 添加 GitHub 身份验证，您必须在 app-config.yaml 文件中配置 GitHub App。

GitHub 身份验证供应商使用以下配置密钥：

- clientId ：您在 GitHub 上生成的客户端 ID。例如： b59241722e3c3b4816e2
- clientSecret ：与生成的客户端 ID 关联的客户端 secret。
- enterpriseInstanceUrl （可选）：GitHub Enterprise 实例的基本 URL。例如： https://ghe.<company>.com。只有 GitHub Enterprise 需要 enterpriseInstanceUrl。
- callbackUrl （可选）：GitHub 在启动 OAuth 流时使用的回调 URL。例如： https://your-intermediate-service.com/handler。只有在 Developer Hub 不是直接接收器时，才需要 callbackUrl，例如当您将一个 OAuth 应用程序用于多个 Developer Hub 实例时。

要配置 GitHub App，请将供应商配置添加到 root auth 配置下的 app-config.yaml 文件中。例如：

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

9.4. 将 GITHUB 供应商添加到 DEVELOPER HUB 前端

要将供应商添加到前端，请将配置中的登录添加到您的 app-config.yaml 文件中。例如：

```
signInPage: github
```

其他资源

- 有关使用 GitHub 验证后台访问的详情，请参考社区文档中的 GitHub 身份验证提供程序。

- 有关将提供程序添加到 Backstage 前端的详情，请参考社区文档中的显示 身份验证。

第 10 章 OPENID CONNECT 身份验证供应商

Red Hat Developer Hub 使用 OpenID Connect (OIDC)身份验证供应商与支持 OIDC 协议的第三方服务进行身份验证。

10.1. 在 DEVELOPER HUB 中使用 OIDC 身份验证提供程序概述

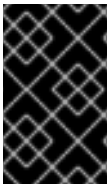
您可以通过在 root auth 配置下更新 app-config.yaml 文件，在 Developer Hub 中配置 OIDC 身份验证供应商。例如：

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

  signInPage: oidc
```

10.2. 使用 OIDC 身份验证供应商配置 KEYCLOAK

Red Hat Developer Hub 包含一个 OIDC 身份验证供应商，可以使用 Keycloak 验证用户。



重要

您在 Keycloak 中创建的用户还必须在 Developer Hub 目录中提供。

流程

1. 在 Keycloak 中，创建一个新域，如 RHDH。

2. 添加新用户。

用户名

用户的用户名，例如： rhdhuser

电子邮件

用户的电子邮件地址。

名

用户的名字。

姓

用户的姓氏。

验证电子邮件

切换到 On。

3. 点 Create。

4. 导航到 Credentials 选项卡。

5. 单击 Set password。

6. 输入 用户帐户的密码，并将 Temporary 切换到 Off。

7. 创建新的客户端 ID，例如 RHDH。

客户端身份验证

切换到 On。

有效的重定向 URI

设置为 OIDC 处理程序 URL，例如 https://<RHDH_URL>/api/auth/oidc/handler/frame。

8. 导航到 Credentials 选项卡，再复制 Client secret。

9. 为下一步保存客户端 ID 和客户端 Secret。

10. 在 Developer Hub 中，在 Developer Hub secret 中添加您的 Keycloak 凭证。

    a. 编辑 Developer Hub secret，如 secrets-rhdh。

    b. 添加以下键/值对：

    AUTH_KEYCLOAK_CLIENT_ID

    输入您在 Keycloak 中生成的客户端 ID，如 RHDH。

    AUTH_KEYCLOAK_CLIENT_SECRET

    输入您在 Keycloak 中生成的客户端 Secret。

11. 在 Developer Hub 自定义配置中设置 OIDC 身份验证供应商。

    a. 编辑自定义 Developer Hub ConfigMap，如 app-config-rhdh。

    b. 在 app-config-rhdh.yaml 内容中，在 root auth 配置下添加 oidc 供应商配置，并为 sign-in 启用 oidc 供应商：

    app-config-rhdh.yaml fragment

```
auth:
  environment: production
  providers:
    oidc:
      production:
      clientId: ${AUTH_KEYCLOAK_CLIENT_ID}
      clientSecret: ${AUTH_KEYCLOAK_CLIENT_SECRET}
```

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

    signInPage: oidc
```

验证

1. 重启 backstage-developer-hub 应用程序以应用更改。

2. 您的 Developer Hub 登录页面显示 使用 OIDC 登录。

10.3. 使用 KEYCLOAK 从 OAUTH2 代理迁移到 DEVELOPER HUB 中的 OIDC

如果您使用 OAuth2 代理作为带有 Keycloak 的身份验证供应商，并且要迁移到 OIDC，您可以更新身份验证供应商配置以使用 OIDC。

流程

1. 在 Keycloak 中，将有效的重定向 URI 更新至 https://<rhdh_url>/api/auth/oidc/handler/frame。确保将 < rhdh_url& gt; 替换为您的 Developer Hub 应用程序 URL，如 my.rhdh.example.com。

2. 将 app-config.yaml 文件的 auth 部分中的 oauth2Proxy 配置值替换为 oidc 配置值。

3. 将 signInPage 配置值从 oauth2Proxy 更新至 oidc。

   以下示例显示 oauth2Proxy 的 auth.providers 和 signInPage 配置，然后再将身份验证供应商迁移到 oidc ：

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

```
  providers:
    oauth2Proxy: {}

signInPage: oauth2Proxy
```

以下示例显示在将身份验证供应商迁移到 oidc 后 auth.providers 和 signInPage 配置：

```
auth:
  environment: production
  session:
    secret: ${SESSION_SECRET}
  providers:
    oidc:
      production:
        metadataUrl: ${KEYCLOAK_METADATA_URL}
        clientId: ${KEYCLOAK_CLIENT_ID}
        clientSecret: ${KEYCLOAK_CLIENT_SECRET}
        prompt: ${KEYCLOAK_PROMPT} # recommended to use auto

signInPage: oidc
```

4. 删除 OAuth2 Proxy sidecar 容器并更新 Helm Chart 的 values.yaml 文件的 upstream.service 部分，如下所示：

- service.ports.backend:7007
- service.ports.targetPort: backend

以下示例显示 oauth2Proxy 的 服务配置，然后再将身份验证提供程序迁移到 oidc ：

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

以下示例显示了将身份验证供应商迁移到 oidc 后的 服务配置 ：

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

5. **升级 Developer Hub Helm Chart。**