Red Hat Developer Hub 1.2

Red Hat Developer Hub 的管理指南

Last Updated: 2024-07-03

法律通告

摘要

Red Hat Developer Hub 是一个企业级平台，用于构建开发人员门户。以管理用户身份，您可以管理其他用户的角色和权限，并配置 Developer Hub 以满足机构的特定需求。

目录

前言

Red Hat Developer Hub 是一个企业级的开放开发人员平台，可用于构建开发人员门户。此平台包含支持和建议的框架，有助于减少开发人员的欺诈和提高其生产力。

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 安装 RED HAT DEVELOPER HUB OPERATOR

作为管理员，您可以安装 Red Hat Developer Hub Operator。授权用户可以使用 Operator 在以下平台上安装 Red Hat Developer Hub：

- Red Hat OpenShift Container Platform (RHOCP)
- Amazon Elastic Kubernetes Service (EKS)
- Microsoft Azure Kubernetes Service (AKS)

RHOCP 目前支持从 4.12 升级到 4.15。另请参阅 Red Hat Developer Hub 生命周期 。

容器可用于以下 CPU 架构：

- AMD64 和 Intel 64 (x86_64)

先决条件

- 以 OpenShift Container Platform Web 控制台的管理员身份登录。
- 您已在项目中配置了适当的角色和权限来创建应用程序。如需更多信息，请参阅有关构建应用程序的 Red Hat OpenShift 文档。

注意

为提高安全性，请在专用的 default 命名空间中部署 Red Hat Developer Hub Operator，如 **rhdh-operator**。集群管理员可以通过角色绑定或集群角色绑定限制其他用户对 Operator 资源的访问。

流程

1. 在 OpenShift Container Platform Web 控制台的 **Administrator** 视角中，进入到 **Operators > OperatorHub**。
2. 在 **Filter by keyword** 框中，输入 Developer Hub 并点 **Red Hat Developer Hub**Operator 卡。
3. 在 **Red Hat Developer Hub Operator**页面中，点 **Install**。
4. 在 **Install Operator** 页面中，使用 **Update channel** 下拉菜单选择您要使用的更新频道：
   - **fast** 通道提供 y-stream (x.y)和 z-stream (x.y.z)更新，例如从 1.1 升级到 1.2，或者从 1.1.0 升级到 1.1.1。

     重要

     fast 通道包括特定版本可用的所有更新。任何更新都可能会在 Red Hat Developer Hub 部署中引入意外更改。参阅发行注记以了解任何可能破坏更改的详细信息。

   - **fast-1.1** 频道只提供 z-stream 更新，例如从 1.1.1 升级到 1.1.2。如果要在以后更新 Red Hat Developer Hub y-version，例如从 1.1 更新至 1.2，您必须手动切换到 **fast** 频道。
5. 在 **Install Operator** 页面中，为 Operator 选择 **Update approval** 策略：
   - 如果选择 **Automatic** 选项，Operator 会更新，而无需手动确认。

- 如果选择 **Manual** 选项，则当更新频道中发布新更新时，会打开通知。在开始安装前，管理员必须手动批准更新。

6. 点 **Install**。

验证

- 要查看已安装的 Red Hat Developer Hub Operator，请点 **View Operator**。

其他资源

- 使用 Operator 在 OpenShift Container Platform 上部署 Red Hat Developer Hub
- 使用 Web 控制台从 OperatorHub 安装

第 2 章 在 OPENSHIFT CONTAINER PLATFORM 上部署 RED HAT DEVELOPER HUB

您可以使用以下方法之一在 OpenShift Container Platform 上安装 Red Hat Developer Hub：

- Helm chart
- Red Hat Developer Hub Operator

2.1. 使用 HELM CHART 在 OPENSHIFT CONTAINER PLATFORM 上部署 RED HAT DEVELOPER HUB

您可以使用 Red Hat OpenShift Container Platform 中的 Helm chart 来安装 Developer Hub，这是灵活的安装方法。

Helm 是 OpenShift Container Platform 上的软件包管理器，它提供以下功能：

- 使用自定义 hook 应用常规应用程序更新
- 管理复杂应用程序的安装
- 提供可在公共和私有服务器上托管的图表
- 支持回滚到以前的应用程序版本

Red Hat Developer Hub Helm chart 位于 OpenShift Dedicated 和 OpenShift Container Platform 的 Helm 目录中。

先决条件

- 已登陆到 OpenShift Container Platform 帐户。
- 具有 OpenShift Container Platform **admin** 角色的用户在项目中配置了适当的角色和权限，以创建应用程序。如需有关 OpenShift Container Platform 角色的更多信息，请参阅使用 RBAC 定义和应用权限。
- 您已在 OpenShift Container Platform 中创建了一个项目。有关在 OpenShift Container Platform 中创建项目的更多信息，请参阅 Red Hat OpenShift Container Platform 文档 。

流程

1. 从 **Developer** Hub web 控制台的 Developer 视角，点 **+Add**。
2. 在 **Developer Catalog** 面板中点 **Helm Chart**。
3. 在 **Filter by keyword** 框中，输入 *Developer Hub* 并点 **Red Hat Developer Hub**卡。
4. 在 Red Hat Developer Hub 页面中，点 **Create**。
5. 从集群中移除，复制 OpenShift Container Platform 路由器主机（例如： **apps.<clusterName>.com**）。
6. 选择单选按钮，以使用表单视图或 YAML 视图来配置 Developer Hub 实例。默认选择 Form view。

- 使用 **Form view**

  a. 要使用 Form 视图配置实例，请进入 **Backstage 实例中的 Root Schema → global → Enable service authentication**，并将 OpenShift Container Platform 路由器主机粘贴到表单上的字段中。

- 使用 **YAML 视图**

  a. 要使用 YAML 视图配置实例，请将 OpenShift Container Platform 路由器主机名粘贴到 **global.clusterRouterBase** 参数值中，如下例所示：

```
global:
  auth:
    backend:
      enabled: true
  clusterRouterBase: apps.<clusterName>.com
  # other Red Hat Developer Hub Helm Chart configurations
```

7. 如果需要，请编辑其他值。

注意

有关主机的信息会被复制，并可以被 Developer Hub 后端访问。

当自动生成 OpenShift Container Platform 路由时，路由的主机值会被推断出来，并将相同的主机信息发送到 Developer Hub。另外，如果自定义域中存在使用值手动设置主机，则自定义主机将具有优先权。

8. 点 **Create** 并等待数据库和 Developer Hub 启动。

9. 点 **Open URL** 图标开始使用 Developer Hub 平台。

注意

如果 Developer Hub 容器无法访问配置文件，则 **developer-hub** pod 可能会处于 **CrashLoopBackOff** 状态。这个错误由以下日志表示：

```
Loaded config from app-config-from-configmap.yaml, env
...
2023-07-24T19:44:46.223Z auth info Configuring "database" as KeyStore provider
type=plugin
Backend failed to start up Error: Missing required config value at
'backend.database.client'
```

要解决这个错误，请验证配置文件。

2.1.1. 使用 Helm CLI 在 Red Hat OpenShift Container Platform 上安装 Red Hat Developer Hub

您可以使用 Helm CLI 在 Red Hat OpenShift Container Platform 上安装 Red Hat Developer Hub。

先决条件

- 已在工作站上安装了 OpenShift CLI (**oc**)。
- 已登陆到 OpenShift Container Platform 帐户。
- 具有 OpenShift Container Platform admin 角色的用户已在项目中配置了适当的角色和权限来创建应用程序。如需有关 OpenShift Container Platform 角色的更多信息，请参阅 [使用 RBAC 来定义和应用权限](https://docs.openshift.com/container-platform/4.15/authentication/using-rbac.html)。
- 您已在 OpenShift Container Platform 中创建了一个项目。有关在 OpenShift Container Platform 中创建项目的更多信息，请参阅 [Red Hat OpenShift Container Platform 文档](https://docs.openshift.com/container-platform/4.15/applications/projects/working-with-projects.html#odc-creating-projects-using-developer-perspective_projects)。
- 已安装 Helm CLI。

流程

1. 创建并激活 < *rhdh*> OpenShift Container Platform 项目：

```
NAMESPACE=<emphasis><rhdh></emphasis>
oc new-project ${NAMESPACE} || oc project ${NAMESPACE}
```

2. 安装 Red Hat Developer Hub Helm Chart:

```
helm upgrade redhat-developer-hub -i https://github.com/openshift-helm-
charts/charts/releases/download/redhat-redhat-developer-hub-1.2.1/redhat-developer-hub-
1.2.1.tgz
```

3. 使用 OpenShift Container Platform 集群中的 Developer Hub 数据库密码和路由器基本 URL 值配置 Developer Hub Helm Chart 实例：

```
PASSWORD=$(oc get secret redhat-developer-hub-postgresql -o jsonpath="
```

```
{.data.password}" | base64 -d)
CLUSTER_ROUTER_BASE=$(oc get route console -n openshift-console -
o=jsonpath='{.spec.host}' | sed 's/^[^.]*\.//')
helm upgrade redhat-developer-hub -i "https://github.com/openshift-helm-
charts/charts/releases/download/redhat-redhat-developer-hub-1.2.1/redhat-developer-hub-
1.2.1.tgz" \
  --set global.clusterRouterBase="$CLUSTER_ROUTER_BASE" \
  --set global.postgresql.auth.password="$PASSWORD"
```

4. 显示正在运行的 Developer Hub 实例 URL：

```
echo "https://redhat-developer-hub-$NAMESPACE.$CLUSTER_ROUTER_BASE"
```

验证

- 在浏览器中打开正在运行的 Developer Hub 实例 URL，以使用 Developer Hub。

2.1.2. 使用 Helm Chart 将自定义应用程序配置文件添加到 OpenShift Container Platform 中

您可以使用 Red Hat Developer Hub Helm Chart 将自定义应用程序配置文件添加到 OpenShift Container Platform 实例中。

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，并根据需要更改该文件。
4. 点 **Create**。
5. 进入 **Helm** 选项卡查看 Helm 发行版本列表。
6. 点击您要使用的 Helm 发行版本的 overflow 菜单，然后选择 **Upgrade**。
7. 使用 **Form view** 或 **YAML** 视图 来编辑 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. 设置 **upstream.backstage.extraAppConfig.configMapRef** 和 **upstream.backstage.extraAppConfig.filename** 参数的值，如下所示：

```
# ... 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**。

2.1.3. 使用 air-gapped 环境中的 Helm Chart 安装 Red Hat Developer Hub

air-gapped 环境（也称为 air-gapped 网络或隔离的网络）通过隔离系统或网络来确保安全性。建立这个隔离以防止 air-gapped 系统和外部源间的未授权访问、数据传输或通信。

您可以在 air-gapped 环境中安装 Red Hat Developer Hub，以确保安全性并满足特定的法规要求。

要在 air-gapped 环境中安装 Developer Hub，您必须有权访问 **registry.redhat.io** 和 air-gapped 环境的 registry。

先决条件

- 已安装 Red Hat OpenShift Container Platform 4.12 或更高版本。
- 您可以访问 **registry.redhat.io**。
- 您可以访问集群的 Red Hat OpenShift Container Platform 镜像 registry。有关公开镜像 registry 的更多信息，请参阅有关 公开 registry 的 Red Hat OpenShift Container Platform 文档。
- 已在工作站上安装了 OpenShift CLI (**oc**)。
- 您已在工作站上安装了 **podman** 命令行工具。
- 您在 Red Hat Developer Portal 中有一个帐户。

流程

1. 运行以下命令，使用 OpenShift CLI (**oc**)登录到 OpenShift Container Platform 帐户：

```
oc login -u <user> -p <password> https://api.<hostname>:6443
```

2. 运行以下命令，使用 **podman** 命令行工具登录到 OpenShift Container Platform 镜像 registry：

```
podman login -u kubeadmin -p $(oc whoami -t) default-route-openshift-image-registry.
<hostname>
```

注意

您可以运行以下命令来获取 OpenShift Container Platform 镜像 registry 的完整主机名，然后使用命令中的主机名登录：

```
REGISTRY_HOST=$(oc get route default-route -n openshift-image-registry --template='{{ .spec.host }}')
```

```
podman login -u kubeadmin -p $(oc whoami -t) $REGISTRY_HOST
```

3. 运行以下命令，登录到 **podman** 中的 **registry.redhat.io** ：

```
podman login registry.redhat.io
```

有关 registry 身份验证的更多信息，请参阅 Red Hat Container Registry 身份验证 。

4. 运行以下命令，将 Developer Hub 和 PostgreSQL 镜像从 Red Hat Image registry 拉取到您的工作站：

```
podman pull registry.redhat.io/rhdh/rhdh-hub-rhel9:{product-chart-version}
```

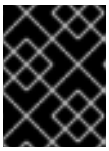
```
podman pull registry.redhat.io/rhel9/postgresql-15:latest
```

5. 运行以下命令，将这两个镜像推送到内部 OpenShift Container Platform 镜像 registry：

```
podman push --remove-signatures registry.redhat.io/rhdh/rhdh-hub-rhel9:{product-chart-version} default-route-openshift-image-registry.<hostname>/<project_name>/rhdh-hub-rhel9:{product-chart-version}
```

```
podman push --remove-signatures registry.redhat.io/rhel9/postgresql-15:latest default-route-openshift-image-registry.<hostname>/<project_name>/postgresql-15:latest
```

有关直接将镜像推送到 OpenShift Container Platform 镜像 registry 的更多信息，请参阅 如何直接将镜像推送到 OpenShift 4 registry 中。



重要

如果发生 x509 错误，请验证您已在 系统上安装了用于 OpenShift Container Platform 路由的 CA 证书。

6. 使用以下命令验证 OpenShift Container Platform 内部 registry 中是否存在这两个镜像：

```
oc get imagestream -n <project_name>
```

7. 运行以下命令，为这两个镜像启用本地镜像查找：

```
oc set image-lookup postgresql-15
```

```
oc set image-lookup  rhdh-hub-rhel9
```

8. 进入 **YAML** 视图，并使用以下值更新 **backstage** 和 **postgresql** 的 **image** 部分：

Developer Hub 镜像的值示例

```
upstream:
  backstage:
    image:
      registry: ""
      repository: rhdh-hub-rhel9
      tag: latest
```

PostgreSQL 镜像的值示例

```
upstream:
  postgresql:
    image:
      registry: ""
      repository: postgresql-15
      tag: latest
```

9. 使用 Helm Chart 安装 Red Hat Developer Hub。有关安装 Developer Hub 的更多信息，请参阅第 2.1 节 “使用 Helm Chart 在 OpenShift Container Platform 上部署 Red Hat Developer Hub”。

2.2. 使用 OPERATOR 在 OPENSHIFT CONTAINER PLATFORM 上部署 RED HAT DEVELOPER HUB

作为开发者，您可以使用 Red Hat OpenShift Container Platform Web 控制台中的 **Developer Catalog** 在 OpenShift Container Platform 上部署 Red Hat Developer Hub 实例。此部署方法使用 Red Hat Developer Hub Operator。

先决条件

- 集群管理员已安装 Red Hat Developer Hub Operator。如需更多信息，请参阅安装 Red Hat Developer Hub Operator。

流程

1. 在 OpenShift Container Platform 中为 Red Hat Developer Hub 实例创建项目，或选择现有项目。

   提示

   有关在 OpenShift Container Platform 中创建项目的更多信息，请参阅 Red Hat OpenShift Container Platform 文档中的使用 Web 控制台 创建项目。

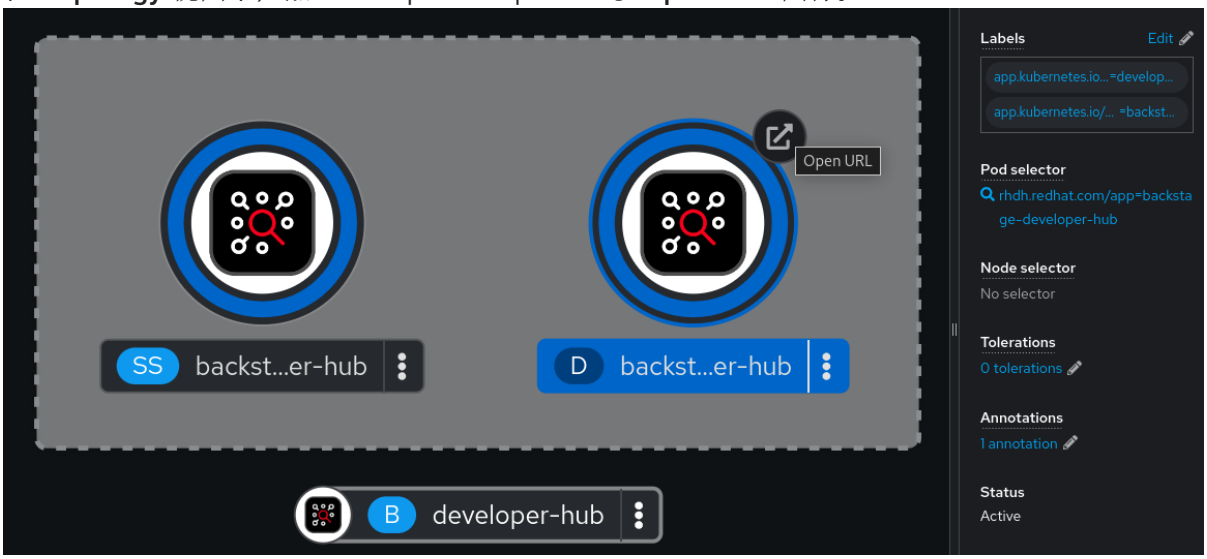
2. 从 OpenShift Container Platform Web 控制台的 **Developer** 视角，点 **+Add**。
3. 在 **Developer Catalog** 面板中，单击 **Operator Backed**。
4. 在 **Filter by keyword** 框中，输入 *Developer Hub* 并点 **Red Hat Developer Hub**卡。
5. 点 **Create**。
6. 为 Red Hat Developer Hub 实例 添加自定义配置。

7. 在 **Create Backstage** 页面中，点 **Create**

验证

pod 就绪后，您可以通过打开 URL 来访问 Red Hat Developer Hub 平台。

1. 点 **Topology** 视图中的 pod 确认 pod 就绪，并确认 **Details** 面板中的 **Status**。当 pod 就绪时，pod 状态为 **Active**。
2. 在 **Topology** 视图中，点 Developer Hub pod 上的 **Open URL** 图标。



其他资源

- OpenShift Container Platform - 构建应用程序概述

2.2.1. 使用 Operator 将自定义应用程序配置文件添加到 OpenShift Container Platform 中

自定义应用程序配置文件是一个 **ConfigMap** 对象，可用于更改 Red Hat Developer Hub 实例的配置。如果要在 Red Hat OpenShift Container Platform 上部署 Developer Hub 实例，您可以使用 Red Hat Developer Hub Operator 将自定义应用程序配置文件添加到 OpenShift Container Platform 实例，方法是创建 **ConfigMap** 对象并在 Developer Hub 自定义资源(CR)中引用它。

自定义应用配置文件包含一个名为 **BACKEND_SECRET** 的敏感环境变量。此变量包含一个强制的后端身份验证密钥，Developer Hub 用来引用 OpenShift Container Platform secret 中定义的环境变量。您必须创建一个名为 'secrets-rhdh' 的 secret，并在 Developer Hub CR 中引用它。

注意

您需要保护 Red Hat Developer Hub 安装不受外部和未授权访问的影响。像管理任何其他机密一样管理后端身份验证密钥。满足强的密码要求，不要在任何配置文件中公开它，仅将其作为环境变量注入配置文件中。

先决条件

- 您有一个活跃的 Red Hat OpenShift Container Platform 帐户。
- 您的管理员已在 OpenShift Container Platform 中安装了 Red Hat Developer Hub Operator。如需更多信息，请参阅安装 Red Hat Developer Hub Operator 。

- 您已在 OpenShift Container Platform 中创建 Red Hat Developer Hub CR。

流程

1. 从 OpenShift Container Platform Web 控制台中的 **Developer** 视角，选择 **Topology** 视图，然后点击 Developer Hub pod 上的 **Open URL** 图标来识别 Developer Hub 外部 URL: *<RHDH_URL>*。
2. 从 OpenShift Container Platform Web 控制台中的 **Developer** 视角，选择 **ConfigMaps** 视图。
3. 点 **Create ConfigMap**。
4. 在 **Configure via** 中选择 **YAML view** 选项，并使用以下示例作为基础模板来创建 **ConfigMap** 对象，如 **app-config-rhdh.yaml** ：

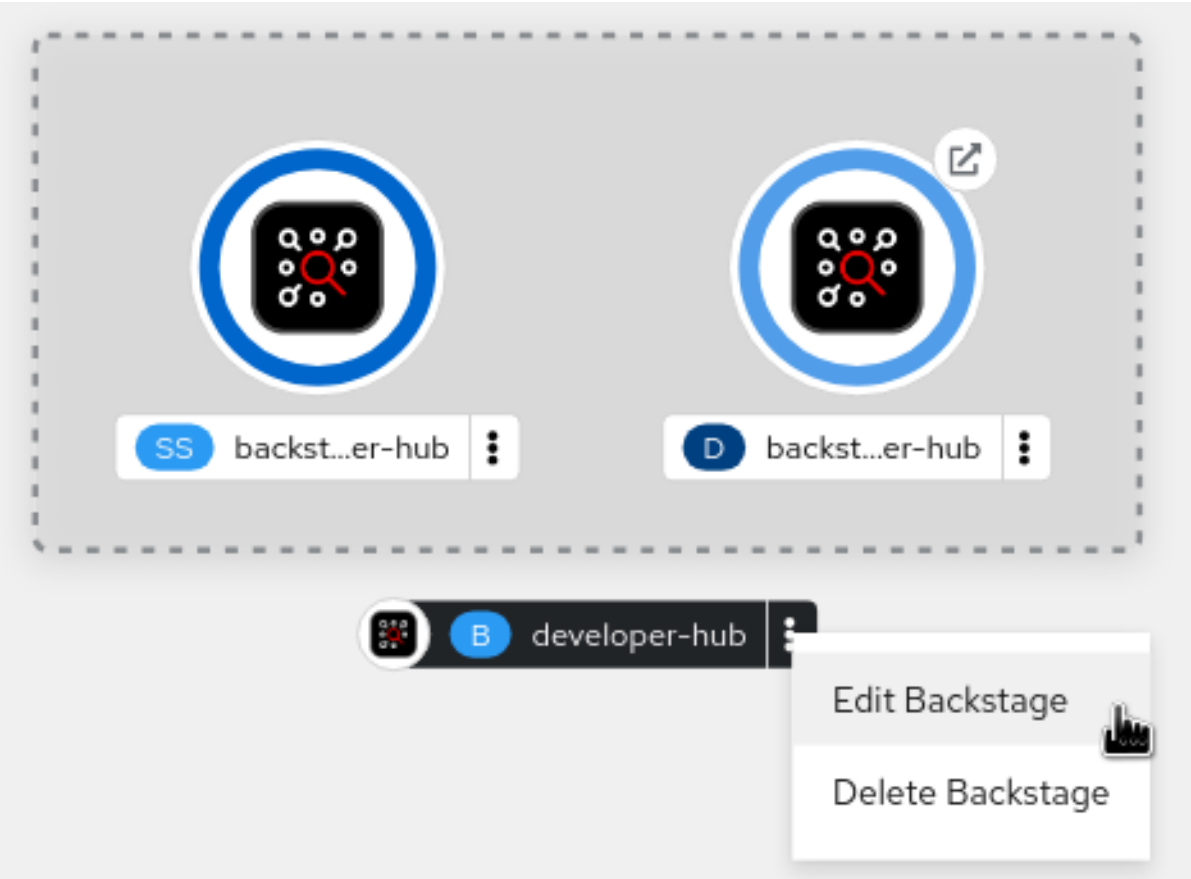
```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app-config-rhdh
data:
  "app-config-rhdh.yaml": |
    app:
      title: Red Hat Developer Hub
      baseUrl: <RHDH_URL> 1
    backend:
      auth:
        keys:
          - secret: "${BACKEND_SECRET}" 2
      baseUrl: <RHDH_URL> 3
      cors:
        origin: <RHDH_URL> 4
```

1. 设置 Red Hat Developer Hub 实例的外部 URL。
2. 使用环境变量公开 OpenShift Container Platform secret，以定义强制的 Developer Hub 后端身份验证密钥。
3. 设置 Red Hat Developer Hub 实例的外部 URL。
4. 设置 Red Hat Developer Hub 实例的外部 URL。

5. 点 **Create**。
6. 选择 **Secrets** 视图。
7. 点 **Create Key/value Secret**。
8. 创建名为 **secrets-rhdh** 的 secret。
9. 添加名为 **BACKEND_SECRET** 的键，并将 base64 编码字符串作为值。为每个 Red Hat Developer Hub 实例使用唯一值。例如，您可以使用以下命令从终端生成密钥：

```
node -p 'require("crypto").randomBytes(24).toString("base64")'
```

10. 点 **Create**。
11. 选择 **Topology** 视图。
12. 点您要使用的 Red Hat Developer Hub 实例的 overflow 菜单，然后选择 **Edit Backstage** 来加载 Red Hat Developer Hub 实例的 YAML 视图。



13. 在 CR 中，输入自定义应用程序配置配置映射的名称作为 **spec.application.appConfig.configMaps** 字段的值，并输入 secret 的名称作为 **spec.application.extraEnvs.secrets** 字段的值。例如：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: example
spec:
  application:
    appConfig:
      mountPath: /opt/app-root/src
      configMaps:
        - name: app-config-rhdh
    extraEnvs:
      secrets:
        - name: secrets-rhdh
    extraFiles:
      mountPath: /opt/app-root/src
    replicas: 1
    route:
      enabled: true
  database:
    enableLocalDb: true
```

14. 点击 **Save**。
15. 返回到 **Topology** 视图，并等待 Red Hat Developer Hub pod 启动。
16. 点 **Open URL** 图标使用 Red Hat Developer Hub 平台以及配置更改。

其他资源

- 如需有关 Developer Hub 中的角色和职责的更多信息，请参阅 Red Hat Developer Hub 中的基于角色的访问控制(RBAC)。

2.2.2. 使用 Red Hat Developer Hub Operator 配置动态插件

您可以将动态插件的配置存储在 **Backstage** 自定义资源(CR)可以引用的 **ConfigMap** 对象中。

注意

如果 **pluginConfig** 字段引用环境变量，您必须在 **secrets-rhdh** secret 中定义变量。

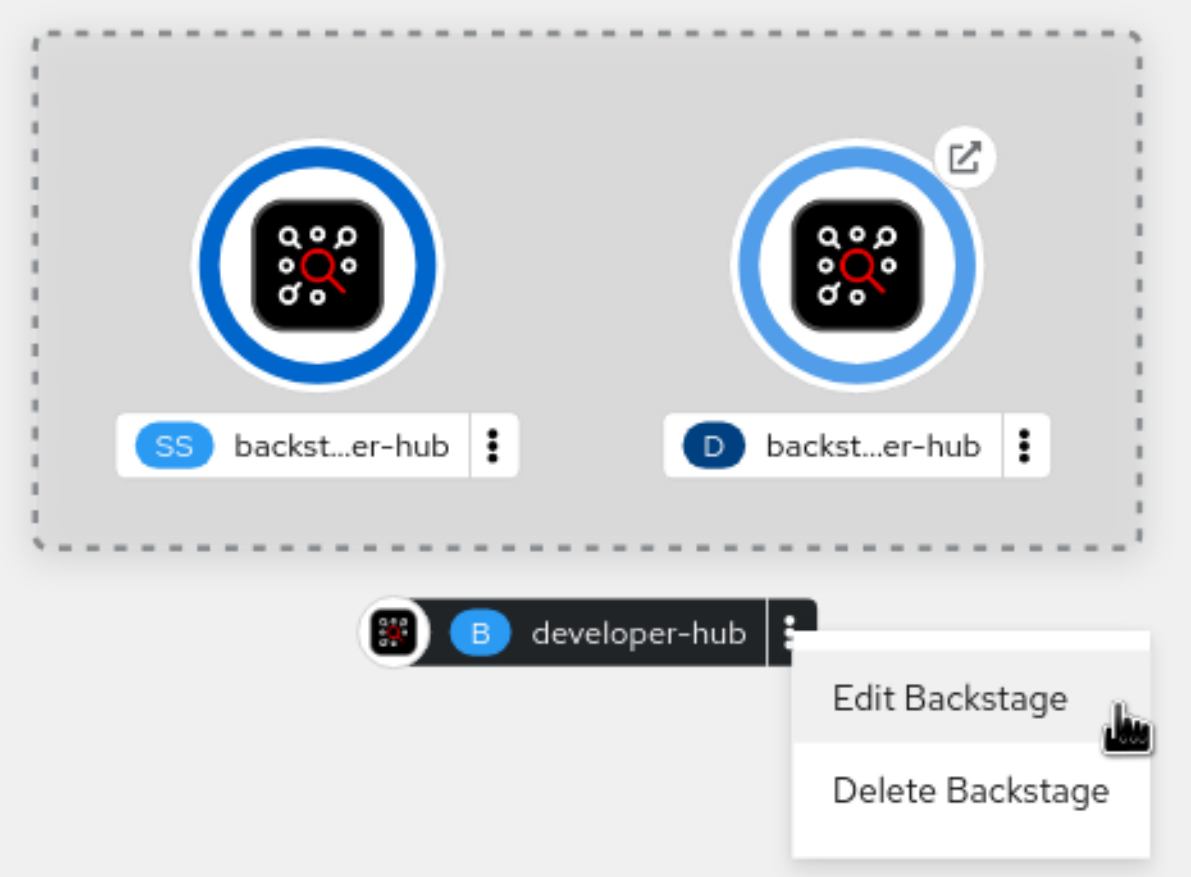
流程

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，根据需要在 **Configure via** 中选择 **YAML view** 选项，并编辑该文件。

   使用 GitHub 动态插件的 ConfigMap 对象示例

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: dynamic-plugins-rhdh
data:
  dynamic-plugins.yaml: |
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: './dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-dynamic'
        disabled: false
        pluginConfig: {}
```

4. 点 **Create**。
5. 进入 **Topology** 视图。
6. 点击您要使用的 Red Hat Developer Hub 实例的 overflow 菜单，然后选择 **Edit Backstage** 来加载 Red Hat Developer Hub 实例的 YAML 视图。



7. 将 **dynamicPluginsConfigMapName** 字段添加到 **Backstage** CR。例如：

```
apiVersion: rhdh.redhat.com/v1alpha1
kind: Backstage
metadata:
  name: my-rhdh
spec:
  application:
# ...
    dynamicPluginsConfigMapName: dynamic-plugins-rhdh
# ...
```

8. 点击 **Save**。

9. 返回到 **Topology** 视图，并等待 Red Hat Developer Hub pod 启动。

10. 点 **Open URL** 图标开始使用带有新配置更改的 Red Hat Developer Hub 平台。

验证

- 通过将 **/api/dynamic-plugins-info/loaded-plugins** 附加到 Red Hat Developer Hub root URL 并检查插件列表来确保已加载了动态插件配置：

  插件列表示例

```
[
  {
    "name": "backstage-plugin-catalog-backend-module-github-dynamic",
    "version": "0.5.2",
    "platform": "node",
```

```
    "role": "backend-plugin-module"
  },
  {
    "name": "backstage-plugin-techdocs",
    "version": "1.10.0",
    "role": "frontend-plugin",
    "platform": "web"
  },
  {
    "name": "backstage-plugin-techdocs-backend-dynamic",
    "version": "1.9.5",
    "platform": "node",
    "role": "backend-plugin"
  },
]
```

2.2.3. 使用 air-gapped 环境中的 Operator 安装 Red Hat Developer Hub

在受限网络上运行的 OpenShift Container Platform 集群中，公共资源不可用。但是，部署 Red Hat Developer Hub Operator 并运行 Developer Hub 需要以下公共资源：

- Operator 镜像(bundle、operator、catalog)
- 操作对象镜像(RHDH, PostgreSQL)

要使这些资源可用，请将它们替换为 OpenShift Container Platform 集群可访问的镜像 registry 中的等效资源。

您可以使用一个帮助程序脚本来镜像所需的镜像，并提供必要的配置，以确保在安装 Red Hat Developer Hub Operator 和创建 Developer Hub 实例时将使用这些镜像。

注意

如果 OpenShift Container Platform 集群已准备好在受限网络中操作，此脚本需要已安装目标镜像 registry。但是，如果您准备集群进行断开连接的使用，您可以使用该脚本在集群中部署镜像 registry，并使用它来镜像(mirror)过程。

先决条件

- 您有一个活跃的 OpenShift CLI (**oc**)会话，其中包含对 OpenShift Container Platform 集群的管理权限。请参阅 OpenShift CLI 入门。
- 您有一个活跃的 **oc registry** 会话到 **registry.redhat.io** 红帽生态系统目录。请参阅 Red Hat Container Registry 身份验证。
- **opm** CLI 工具已安装。请参阅安装 opm CLI。
- 已安装 jq 软件包。请参阅 下载 jq。
- podman 已安装。请参阅 Podman 安装说明。
- 已安装 Skopeo 版本 1.14 或更高版本。请参阅 安装 Skopeo。
- 如果您已经有一个集群的镜像 registry，则需要一个活跃的 Skopeo 会话，并具有对此 registry 的管理访问权限。 请参阅为断开连接的安装授权 registry 和镜像镜像。

注意

内部 OpenShift Container Platform 集群镜像 registry 不能用作目标镜像 registry。请参阅关于镜像 registry。

- 如果要创建自己的镜像 registry，请参阅为 Red Hat OpenShift 创建带有镜像 registry 的镜像 registry。
- 如果您还没有镜像 registry，您可以使用 helper 脚本为您创建一个，您需要以下额外先决条件：
  - 已安装 cURL 软件包。对于 Red Hat Enterprise Linux，可通过安装 curl 软件包来使用 curl 命令。要将 curl 用于其他平台，请查看 cURL 网站。
  - **htpasswd** 命令可用。对于 Red Hat Enterprise Linux，可以通过安装 **httpd-tools** 软件包来使用 **htpasswd** 命令。

流程

1. 下载并运行镜像脚本以安装自定义 Operator 目录并镜像相关的镜像：**prepare-restricted-environment.sh** (source)。

```
curl -sSLO https://raw.githubusercontent.com/janus-idp/operator/1.1.x/.rhdh/scripts/prepare-restricted-environment.sh

# if you do not already have a target mirror registry
# and want the script to create one for you
# use the following example:
bash prepare-restricted-environment.sh \
  --prod_operator_index "registry.redhat.io/redhat/redhat-operator-index:v4.14" \
  --prod_operator_package_name "rhdh" \
  --prod_operator_bundle_name "rhdh-operator" \
  --prod_operator_version "v1.1.1"

# if you already have a target mirror registry
# use the following example:
bash prepare-restricted-environment.sh \
  --prod_operator_index "registry.redhat.io/redhat/redhat-operator-index:v4.14" \
  --prod_operator_package_name "rhdh" \
  --prod_operator_bundle_name "rhdh-operator" \
  --prod_operator_version "v1.1.1" \
  --use_existing_mirror_registry "my_registry"
```

注意

完成该脚本可能需要几分钟时间，因为它会将多个镜像复制到镜像 registry。

第 3 章 配置外部 POSTGRESQL 数据库

作为管理员，您可以在 Red Hat Developer Hub 中配置和使用外部 PostgreSQL 数据库。您可以使用 PostgreSQL 证书文件使用 Operator 或 Helm Chart 配置外部 PostgreSQL 实例。

注意

Developer Hub 仅支持配置外部 PostgreSQL 数据库。您可以执行维护活动，如备份数据或为外部 PostgreSQL 数据库配置高可用性(HA)。

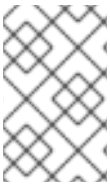
另外，使用 Red Hat Developer Hub Operator 或 Helm Chart 配置外部 PostgreSQL 实例并不适用于生产环境。

3.1. 使用 OPERATOR 配置外部 POSTGRESQL 实例

您可以使用 Red Hat Developer Hub Operator 配置外部 PostgreSQL 实例。默认情况下，Operator 会在部署 RHDH 实例的同一命名空间中创建和管理一个 PostgreSQL 本地实例。但是，您可以更改此默认设置来配置外部 PostgreSQL 数据库服务器，例如 Amazon Web Services (AWS)相关数据库服务(RDS)或 Azure 数据库。

先决条件

- 您使用受支持的 PostgreSQL 版本。如需更多信息，请参阅 产品生命周期页面。
- 您有以下详情：
  - **db-host** ：注意 PostgreSQL 实例域名系统(DNS)或 IP 地址
  - **db-port** ：注意 PostgreSQL 实例端口号，如 **5432**
  - **用户名** ：注意要连接到 PostgreSQL 实例的用户名
  - **密码** ：注意连接到 PostgreSQL 实例的密码
- 已安装 Red Hat Developer Hub Operator。
- 可选： 您有一个 CA 证书、传输层安全(TLS)私钥和 TLS 证书，以便您可以使用 TLS 协议保护数据库连接。如需更多信息，请参阅 PostgreSQL 供应商文档。



注意

默认情况下，Developer Hub 为每个插件使用一个数据库，并在未找到时自动创建它。除了 **PSQL Database** 特权外，您可能需要 **Create Database** 特权来配置外部 PostgreSQL 实例。

流程

1. 可选：创建一个证书 secret 来使用 TLS 连接配置 PostgreSQL 实例：

```
cat <<EOF | oc -n <your-namespace> create -f -
apiVersion: v1
kind: Secret
metadata:
 name: <crt-secret> 1
```

```
type: Opaque
stringData:
 postgres-ca.pem: |-
  -----BEGIN CERTIFICATE-----
  <ca-certificate-key> 2
 postgres-key.key: |-
  -----BEGIN CERTIFICATE-----
  <tls-private-key> 3
 postgres-crt.pem: |-
  -----BEGIN CERTIFICATE-----
  <tls-certificate-key> 4
  # ...
EOF
```

1. 提供证书 secret 的名称。
2. 提供 CA 证书密钥。
3. 可选：提供 TLS 私钥。
4. 可选：提供 TLS 证书密钥。

2. 创建凭证 secret 以与 PostgreSQL 实例连接：

```
cat <<EOF | oc -n <your-namespace> create -f -
apiVersion: v1
kind: Secret
metadata:
 name: <cred-secret> 1
type: Opaque
stringData: 2
 POSTGRES_PASSWORD: <password>
 POSTGRES_PORT: "<db-port>"
 POSTGRES_USER: <username>
 POSTGRES_HOST: <db-host>
 PGSSLMODE: <ssl-mode> # for TLS connection 3
 NODE_EXTRA_CA_CERTS: <abs-path-to-pem-file> # for TLS connection, e.g. /opt/app-root/src/postgres-crt.pem 4
EOF
```

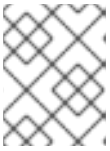
1. 提供凭证 secret 的名称。
2. 提供用于连接 PostgreSQL 实例的凭证数据。
3. 可选：根据所需的 安全套接字层(SSL)模式提供值。
4. 可选： 只有在需要 PostgreSQL 实例的 TLS 连接时才提供值。

3. 创建 **Backstage** 自定义资源(CR)：

```
cat <<EOF | oc -n <your-namespace> create -f -
apiVersion: rhdh.redhat.com/v1alpha1
kind: Backstage
```

```
metadata:
  name: <backstage-instance-name>
spec:
  database:
    enableLocalDb: false (1)
  application:
    extraFiles:
      mountPath: <path> # e g /opt/app-root/src
      secrets:
        - name: <crt-secret> (2)
          key: postgres-crt.pem, postgres-ca.pem, postgres-key.key # key name as in <crt-secret> Secret
    extraEnvs:
      secrets:
        - name: <cred-secret> (3)
          # ...
```

1. 将 **enableLocalDb** 参数的值设置为 **false**，以禁用创建本地 PostgreSQL 实例。
2. 如果您配置了 TLS 连接，请提供证书 secret 的名称。
3. 提供您创建的凭证 secret 的名称。



注意

Backstage CR 中列出的环境变量与 Operator 默认配置一起工作。如果更改了 Operator 默认配置，您必须相应地重新配置 **Backstage** CR。

4. 将 **Backstage** CR 应用到已部署 RHDH 实例的命名空间。

3.2. 使用 HELM CHART 配置外部 POSTGRESQL 实例

您可以使用 Helm Chart 配置外部 PostgreSQL 实例。默认情况下，Helm Chart 会在部署 RHDH 实例的同一命名空间中创建和管理 PostgreSQL 本地实例。但是，您可以更改此默认设置来配置外部 PostgreSQL 数据库服务器，例如 Amazon Web Services (AWS)相关数据库服务(RDS)或 Azure 数据库。

先决条件

- 您使用受支持的 PostgreSQL 版本。如需更多信息，请参阅 产品生命周期页面。
- 您有以下详情：
  - **db-host** ：注意 PostgreSQL 实例域名系统(DNS)或 IP 地址
  - **db-port** ：注意 PostgreSQL 实例端口号，如 **5432**
  - **用户名** ：注意要连接到 PostgreSQL 实例的用户名
  - **密码** ：注意连接到 PostgreSQL 实例的密码
- 已使用 Helm Chart 安装 RHDH 应用程序。
- 可选： 您有一个 CA 证书、传输层安全(TLS)私钥和 TLS 证书，以便您可以使用 TLS 协议保护数据库连接。如需更多信息，请参阅 PostgreSQL 供应商文档。

注意

默认情况下，Developer Hub 为每个插件使用一个数据库，并在未找到时自动创建它。除了 **PSQL Database** 特权外，您可能需要 **Create Database** 特权来配置外部 PostgreSQL 实例。

流程

1. 可选：创建一个证书 secret 来使用 TLS 连接配置 PostgreSQL 实例：

```
cat <<EOF | oc -n <your-namespace> create -f -
apiVersion: v1
kind: Secret
metadata:
 name: <crt-secret> (1)
type: Opaque
stringData:
 postgres-ca.pem: |-
  -----BEGIN CERTIFICATE-----
  <ca-certificate-key> (2)
 postgres-key.key: |-
  -----BEGIN CERTIFICATE-----
  <tls-private-key> (3)
 postgres-crt.pem: |-
  -----BEGIN CERTIFICATE-----
  <tls-certificate-key> (4)
  # ...
EOF
```

(1) 提供证书 secret 的名称。

(2) 提供 CA 证书密钥。

(3) 可选：提供 TLS 私钥。

(4) 可选：提供 TLS 证书密钥。

2. 创建凭证 secret 以与 PostgreSQL 实例连接：

```
cat <<EOF | oc -n <your-namespace> create -f -
apiVersion: v1
kind: Secret
metadata:
 name: <cred-secret> (1)
type: Opaque
stringData: (2)
 POSTGRES_PASSWORD: <password>
 POSTGRES_PORT: "<db-port>"
 POSTGRES_USER: <username>
 POSTGRES_HOST: <db-host>
 PGSSLMODE: <ssl-mode> # for TLS connection (3)
 NODE_EXTRA_CA_CERTS: <abs-path-to-pem-file> # for TLS connection, e.g. /opt/app-root/src/postgres-crt.pem (4)
EOF
```

–

1. 提供凭证 secret 的名称。
2. 提供用于连接 PostgreSQL 实例的凭证数据。
3. 可选：根据所需的 安全套接字层(SSL)模式提供值。
4. 可选： 只有在需要 PostgreSQL 实例的 TLS 连接时才提供值。

3. 在名为 **values.yaml** 的 Helm 配置文件中配置 PostgreSQL 实例：

```
# ...
upstream:
  postgresql:
    enabled: false  # disable PostgreSQL instance creation (1)
    auth:
      existingSecret: <cred-secret> # inject credentials secret to Backstage (2)
  backstage:
    appConfig:
      backend:
        database:
          connection:  # configure Backstage DB connection parameters
            host: ${POSTGRES_HOST}
            port: ${POSTGRES_PORT}
            user: ${POSTGRES_USER}
            password: ${POSTGRES_PASSWORD}
            ssl:
              rejectUnauthorized: true,
              ca:
                $file: /opt/app-root/src/postgres-ca.pem
              key:
                $file: /opt/app-root/src/postgres-key.key
              cert:
                $file: /opt/app-root/src/postgres-crt.pem
    extraEnvVarsSecrets:
      - <cred-secret> # inject credentials secret to Backstage (3)
    extraEnvVars:
      - name: BACKEND_SECRET
        valueFrom:
          secretKeyRef:
            key: backend-secret
            name: '{{ include "janus-idp.backend-secret-name" $ }}'
    extraVolumeMounts:
      - mountPath: /opt/app-root/src/dynamic-plugins-root
        name: dynamic-plugins-root
      - mountPath: /opt/app-root/src/postgres-crt.pem
        name: postgres-crt # inject TLS certificate to Backstage cont. (4)
        subPath: postgres-crt.pem
      - mountPath: /opt/app-root/src/postgres-ca.pem
        name: postgres-ca # inject CA certificate to Backstage cont. (5)
        subPath: postgres-ca.pem
      - mountPath: /opt/app-root/src/postgres-key.key
        name: postgres-key # inject TLS private key to Backstage cont. (6)
        subPath: postgres-key.key
    extraVolumes:
```

```
- ephemeral:
    volumeClaimTemplate:
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 1Gi
  name: dynamic-plugins-root
- configMap:
    defaultMode: 420
    name: dynamic-plugins
    optional: true
  name: dynamic-plugins
- name: dynamic-plugins-npmrc
  secret:
    defaultMode: 420
    optional: true
    secretName: dynamic-plugins-npmrc
- name: postgres-crt
  secret:
    secretName: <crt-secret> 7
    # ...
```

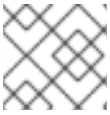
1. 将 **upstream.postgresql.enabled** 参数的值设置为 **false**，以禁用创建本地 PostgreSQL 实例。
2. 提供凭证 secret 的名称。
3. 提供凭证 secret 的名称。
4. 可选：仅为 TLS 连接提供 TLS 证书的名称。
5. 可选：仅为 TLS 连接提供 CA 证书的名称。
6. 可选：仅在 TLS 连接需要私钥时提供 TLS 私钥的名称。
7. 如果您配置了 TLS 连接，请提供证书 secret 的名称。

4. 在名为 **values.yaml** 的 Helm 配置文件中应用配置更改：

```
helm upgrade -n <your-namespace> <your-deploy-name> openshift-helm-charts/redhat-developer-hub -f values.yaml --version 1.2.1
```

3.3. 使用 OPERATOR 将本地数据库迁移到外部数据库服务器

默认情况下，Red Hat Developer Hub 在 PostgreSQL 数据库中托管每个插件的数据。当您获取数据库列表时，您可能会看到基于 Developer Hub 中配置的插件数量的多个数据库。您可以将托管在本地 PostgreSQL 服务器上的 RHDH 实例迁移到外部 PostgreSQL 服务，如 AWS RDS、Azure 数据库或 Crunchy 数据库。要从每个 RHDH 实例迁移数据，您可以使用 PostgreSQL 工具，如 **pg_dump** 和 **psql** 或 **pgAdmin**。



注意

以下流程使用 数据库复制脚本 进行快速迁移。

先决条件

- 您已在本地机器上安装了 **pg_dump** 和 **psql** 工具。
- 对于数据导出，您有 PGSQL 用户特权，以便对本地数据库进行完整的转储。
- 对于数据导入，您有 PGSQL admin 特权来创建外部数据库，并使用数据库转储填充。

流程

1. 在终端中运行以下命令，为本地 PostgreSQL 数据库 pod 配置端口转发：

```
oc port-forward -n <your-namespace> <pgsql-pod-name> <forward-to-port>:<forward-from-port>
```

其中：

- & **lt;pgsql-pod-name&** gt; 变量表示 PostgreSQL pod 的名称，格式为 **backstage-psql-<deployment-name>-<_index>**。
- & **lt;forward-to-port** > 变量表示您选择的端口将 PostgreSQL 数据转发到。
- & **lt;forward-from-port** > 变量表示本地 PostgreSQL 实例端口，如 **5432**。

示例：配置端口转发

```
oc port-forward -n developer-hub backstage-psql-developer-hub-0 15432:5432
```

2. 复制以下 **db_copy.sh** 脚本，并根据您的配置编辑详情：

```
#!/bin/bash

to_host=<db-service-host> 1
to_port=5432 2
to_user=postgres 3

from_host=127.0.0.1 4
from_port=15432 5
from_user=postgres 6

allDB=("backstage_plugin_app" "backstage_plugin_auth" "backstage_plugin_catalog" "backstage_plugin_permission" "backstage_plugin_scaffolder" "backstage_plugin_search") 7

for db in ${!allDB[@]};
do
  db=${allDB[$db]}
  echo Copying database: $db
  PGPASSWORD=$TO_PSW psql -h $to_host -p $to_port -U $to_user -c "create database $db;"
```

```
  pg_dump -h $from_host -p $from_port -U $from_user -d $db | PGPASSWORD=$TO_PSW
psql -h $to_host -p $to_port -U $to_user -d $db
done
```

1. 目标主机名，例如 < **db-instance-name>.rds.amazonaws.com**。
2. 目的地端口，如 **5432**。
3. 目标服务器用户名，例如 **postgres**。
4. 源主机名，如 **127.0.0.1**。
5. 源端口号，如 < **forward-to-port>** 变量。
6. 源服务器用户名，如 **postgres**。
7. 要使用双引号导入的数据库名称，例如："backstage_plugin_ **app" "backstage_plugin_auth" "backstage_plugin_catalog" "backstage_plugin_permission" "backstage_plugin_permission" "backstage_plugin_scaffolder" "backstage_plugin_search"**）。

3. 创建用于复制数据的目的地数据库：

```
/bin/bash TO_PSW=<destination-db-password> /path/to/db_copy.sh (1)
```

1. & **lt;destination-db-password** > 变量表示要连接到目标数据库的密码。

注意

当数据复制完成后，您可以停止端口转发。有关处理大型数据库和使用压缩工具的更多信息，请参阅 PostgreSQL 网站上的 Handling Large Databases 部分。

4. 重新配置 **Backstage** 自定义资源(CR)。如需更多信息，请参阅使用 Operator 配置外部 PostgreSQL 实例。

5. 在重新配置后，检查 **Backstage** CR 的末尾是否存在以下代码：

```
# ...
spec:
  database:
    enableLocalDb: false
  application:
  # ...
    extraFiles:
      secrets:
        - name: <crt-secret>
          key: postgres-crt.pem # key name as in <crt-secret> Secret
    extraEnvs:
      secrets:
        - name: <cred-secret>
# ...
```

注意

重新配置 **Backstage** CR 会删除对应的 **StatefulSet** 和 **Pod** 对象，但不删除 **PersistenceVolumeClaim** 对象。使用以下命令删除本地 **PersistenceVolumeClaim** 对象：

```
oc -n developer-hub delete pvc <local-psql-pvc-name>
```

其中，&lt ;**local-psql-pvc-name&gt**; 变量采用 **data-<psql-pod-name>** 格式。

6. 应用配置更改。

验证

1. 运行以下命令，验证您的 RHDH 实例是否使用迁移的数据运行，且不包含本地 PostgreSQL 数据库：

```
oc get pods -n <your-namespace>
```

2. 检查以下详情的输出：

- **backstage-developer-hub-xxx** pod 处于 running 状态。
- **backstage-psql-developer-hub-0** pod 不可用。
  您还可以使用 OpenShift Container Platform Web 控制台中的 **Topology** 视图来验证这些详情。

第 4 章 在 RED HAT DEVELOPER HUB 中启用身份验证

Red Hat Developer Hub 中的身份验证有助于用户登录、识别和访问外部资源。它支持多个身份验证提供程序。

身份验证供应商通常使用以下方法：

- 一个用于登录和识别的供应商。
- 用于访问外部资源的其他提供程序。

Red Hat Developer Hub 支持以下身份验证供应商：

Microsoft Azure
microsoft
GitHub
github
Keycloak
OIDC

对于您要使用的每个供应商，请按照专用流程完成以下任务：

1. 设置身份验证供应商和 Red Hat Developer Hub 需要通信的共享 secret。
2. 将 Red Hat Developer Hub 配置为使用身份验证供应商。

4.1. 启用 MICROSOFT AZURE 身份验证供应商

Red Hat Developer Hub 包括一个 Microsoft Azure 身份验证供应商，可以使用 OAuth 验证用户。

流程

1. 要允许 Developer Hub 与 Microsoft Azure 进行身份验证，请在 Microsoft Azure 中创建 OAuth 应用程序。
   a. 进入 Azure Portal > App registrations，并为 Developer Hub 创建 **App Registration**。
   b. 在 **App registration** overview 页面中，使用 **配置添加新的 Web 平台** 配置：

      重定向 URI
      输入 Developer Hub 中设置的后端身份验证 URI： **https:// *<APP_FQDN>*; /api/auth/microsoft/handler/frame**

      front-channel logout URL
      留空。

      隐式授权和混合流
      保留所有复选框都未清除。

   c. 在 **API 权限** 选项卡中，单击 **Add Permission**，然后为 **Microsoft Graph API** 添加以下 **委派权限** ：

      email, offline_access, openid, 配置集, user.Read, （可选）

您可以在此处和 Developer Hub 配置中定义 Microsoft Graph API 的可选自定义范围(**app-config-rhdh.yaml**)。

注意

您的公司可能需要您授予 admin 同意这些权限。即使您的公司不需要管理员同意，您可能也要这样做，因为这意味着用户在第一次访问阶段时不需要单独同意。要授予 admin 同意，目录 admin 必须前往 admin consent 页面，然后单击 **COMPANY NAME 的 Grant admin consent**。

a. 进入 **Certificates & Secrets** 页面，然后进入 **Client secret** 选项卡，并创建新的客户端 secret。为下一步保存 **Client secret**。

1. 在 Developer Hub secret 中添加 Microsoft Azure 凭证。

b. 编辑 Developer Hub secret，如 **secrets-rhdh**。

c. 添加以下键/值对：

- **AUTH_AZURE_CLIENT_ID** ：输入您在 Microsoft Azure 上生成的 **应用程序 ID**。
- **AUTH_AZURE_CLIENT_SECRET** ：输入您在 Microsoft Azure 上生成的 **客户端 secret**。
- **AUTH_AZURE_TENANT_ID**: 在 Microsoft Azure 上输入您的 **租户 ID**。

1. 在 Developer Hub 自定义配置中设置 Microsoft Azure 身份验证供应商。编辑自定义 Developer Hub 配置映射，如 **app-config-rhdh**。

   在 **app-config-rhdh.yaml** 内容中，在 root **auth** 配置下添加 **microsoft** 供应商配置，并启用 **microsoft** 供应商 for sign-in ：

   app-config-rhdh.yaml fragment

```
auth:
  environment: production
  providers:
    microsoft:
      production:
        clientId: ${AUTH_AZURE_CLIENT_ID}
        clientSecret: ${AUTH_AZURE_CLIENT_SECRET}
        tenantId: ${AUTH_AZURE_TENANT_ID}
        # domainHint: ${AUTH_AZURE_TENANT_ID} 1
        # additionalScopes: 2
          # - Mail.Send
signInPage: microsoft 3
```

1. 可选的用于单租户应用程序。您可以通过从其他租户过滤出帐户，减少具有多个租户中帐户的登录侵权。如果要将此参数用于单租户应用，请取消注释并输入租户 ID。如果您的应用程序注册是多租户的，请将此参数留空。如需更多信息，请参阅 Home Realm Discovery。
2. 可选用于其他范围。要为应用程序注册添加范围，取消注释并输入您要添加的范围列表。默认值为 **['user.read']**。
3. 要启用 Microsoft Azure 供应商作为默认登录供应商。

注意

对于对外向访问有限制的环境，可选，如防火墙规则。如果您的环境有传出的访问限制，请确保您的 Backstage 后端可以访问以下主机：

- **login.microsoftonline.com** ：要获取并交换授权代码和访问令牌。
- **graph.microsoft.com** ：要获取用户配置文件信息（如此源代码中所示）。如果这个主机无法访问，用户可能会看到 *Authentication failed，在尝试登录时无法获取用户配置集* 错误。

4.2. 启用 GITLAB OAUTH 身份验证供应商

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

先决条件

- 已使用自定义配置映射和 secret 配置 Developer Hub。

流程

1. 要允许 Developer Hub 通过 Gitlab 进行身份验证，请在 Gitlab 中创建 OAuth 应用。
进入 GitLab User settings > Applications，然后点击 **Add new application** 按钮。

   Name

   输入应用程序名称，如 **Developer Hub**。

   重定向 URI

   输入 Developer Hub 中设置的后端身份验证 URI，如 **http://*<APP_FQDN>*/api/auth/gitlab/handler/frame**。由于 GitLab OAuth 的 criuuliarity，请确保 URL 在 'frame' 后没有结尾的 /。

   范围

   从列表中选择以下范围并点 **Save application** ：

   read_user

   通过 /user API 端点授予经过身份验证的用户配置集的只读访问权限，其中包括用户名、公共电子邮件和全名。另外，还授予对 /users 下的只读 API 端点的访问权限。

   read_repository

   通过 /user API 端点授予经过身份验证的用户配置集的只读访问权限，其中包括用户名、公共电子邮件和全名。另外，还授予对 /users 下的只读 API 端点的访问权限。

   write_repository

   使用 Git-over-HTTP （不使用 API），授予对私有项目上的存储库的读取/写入访问权限。

   openid

   授予使用 OpenID Connect 通过 GitLab 进行身份验证的权限。另外，也授予用户配置文件和组成员资格的只读访问权限。

   配置集

   使用 OpenID Connect 授予用户配置集数据的只读访问权限。

   email

   使用 OpenID Connect 授予用户主电子邮件地址的只读访问权限。
   为下一步保存 **应用程序 ID** 和 **Secret**。

2. 在 Developer Hub secret 中添加 Gitlab 凭据。

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

   b. 添加以下键/值对：

   AUTH_GITLAB_CLIENT_ID

   输入您在 GitLab **上生成的应用程序 ID**，如 **4928c033ab3d592845c044a653bc20583baf84f2e67b954c6fdb32a532ab76c9**。

   AUTH_GITLAB_CLIENT_SECRET

   输入您在 Gitlab 上生成的 **Secret**，如 **gloas-f2c9c350759cc08346fbf94a476ae83c579c76dd629fc5eeef9dc21eedfe0475**。

3. 在 Developer Hub 自定义配置中设置 Gitlab 身份验证提供程序。

   a. 编辑自定义 Developer Hub 配置映射，如 **app-config-rhdh**。

   b. 在 **app-config-rhdh.yaml** 内容中，将 **gitlab** 提供程序配置添加到 root **auth** 配置下，并为 sign-in 启用 **gitlab** 提供程序：

   app-config-rhdh.yaml fragment

```
auth:
  environment: production
  providers:
    gitlab:
      production:
        clientId: ${AUTH_GITLAB_CLIENT_ID}
        clientSecret: ${AUTH_GITLAB_CLIENT_SECRET}
        # audience: https://gitlab.company.com (1)
        # callbackUrl: https://<APP_FQDN>/api/auth/gitlab/handler/frame (2)
signInPage: gitlab (3)
```

   (1) 另外，在使用自托管的 Gitlab: 取消注释时，并输入 GitLab 实例基本 URL，如 **https://gitlab.company.com**。

   (2) 另外，在使用自定义重定向 URI 时，输入创建 GitLab OAuth App 时注册的 **Redirect URI** 的 URL，如 **http:// *<APP_FQDN>* /api/auth/gitlab/handler/frame**。由于 GitLab OAuth 的 criuuliarity，请确保 URL 在 'frame' 后没有结尾的 /。

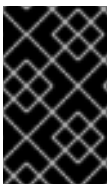
   (3) 将 Gitlab 提供程序启用为默认登录提供程序。

验证

1. **backstage-developer-hub** 部署使用更新的配置启动 pod。
2. 您的 Developer Hub 登录页面显示 **使用 GitLab 的 Sign in**。

第 5 章 TELEMETRY 数据收集

遥测数据收集功能默认是启用的。Red Hat Developer Hub 使用 **backstage-plugin-analytics-provider-segment** 插件向红帽发送遥测数据。



重要

您可以根据您的需要禁用遥测数据收集功能。例如，在 air-gapped 环境中，您可以禁用此功能以避免不必要的出站请求影响 RHDH 应用程序的响应。如需了解更多详细信息，请参阅 RHDH 中的 禁用遥测数据收集 部分。

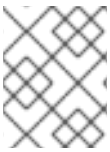
红帽收集并分析以下数据以改进 Red Hat Developer Hub 体验：

- 页面事件访问并点击链接或按钮。
- 系统相关信息，如 locale, timezone, user agent，包括浏览器和操作系统详情。
- 与页面相关的信息，如标题、类别、扩展名称、URL、路径、引用器和搜索参数。
- 匿名 IP 地址，记录为 **0.0.0.0**。
- 匿名用户名哈希，这是唯一标识符，仅用于识别 RHDH 应用的唯一用户数量。

通过 RHDH，您可以根据您的需要自定义遥测数据收集功能和遥测 Segment 源配置。

5.1. 在 RHDH 中禁用遥测数据收集

要禁用遥测数据收集，您必须使用 Helm Chart 或 Red Hat Developer Hub Operator 配置禁用 **analytics-provider-segment** 插件。



注意

如果您的动态插件配置中已存在 **analytics-provider-segment** 插件，请将 **plugins.disabled** 参数的值设置为 **true** 以禁用遥测数据收集。

流程

1. 使用以下选项之一禁用 **analytics-provider-segment** 插件：

使用 Helm Chart

- 在 Helm 配置文件中添加以下 YAML 代码：

```
# ...
global:
  dynamic:
    plugins:
      - package: './dynamic-plugins/dist/janus-idp-backstage-plugin-analytics-provider-segment'
        disabled: true
# ...
```

使用 Operator

a. 执行以下步骤之一：

- 如果您已创建了 **dynamic-plugins-rhdh** ConfigMap 文件，请将 **analytics-provider-segment** 插件添加到插件列表中，并将其 **plugins.disabled** 参数设置为 **true**。
- 如果您还没有创建 ConfigMap 文件，请使用以下 YAML 代码创建该文件：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: dynamic-plugins-rhdh
data:
  dynamic-plugins.yaml: |
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: './dynamic-plugins/dist/janus-idp-backstage-plugin-analytics-provider-segment'
        disabled: true
```

b. 将 **dynamicPluginsConfigMapName** 参数的值设置为 **Backstage** 自定义资源中的 ConfigMap 文件的名称：

```
# ...
spec:
  application:
    dynamicPluginsConfigMapName: dynamic-plugins-rhdh
# ...
```

2. 保存配置更改。

5.2. 在 RHDH 中启用遥测数据收集

遥测数据收集功能默认是启用的。但是，如果您禁用了这个功能并希望重新启用它，则必须使用 Helm Chart 或 Red Hat Developer Hub Operator 配置启用 **analytics-provider-segment** 插件。

注意

如果您的动态插件配置中已存在 **analytics-provider-segment** 插件，请将 **plugins.disabled** 参数的值设置为 **false** 以启用遥测数据收集。

流程

1. 使用以下选项之一配置 **analytics-provider-segment** 插件：

使用 Helm Chart

- 在 Helm 配置文件中添加以下 YAML 代码：

```
# ...
global:
  dynamic:
    plugins:
      - package: './dynamic-plugins/dist/janus-idp-backstage-plugin-analytics-provider-
```

```
segment'
      disabled: false
# ...
```

使用 Operator

a. 执行以下步骤之一：

- 如果您已创建了 **dynamic-plugins-rhdh** ConfigMap 文件，请将 **analytics-provider-segment** 插件添加到插件列表中，并将其 **plugins.disabled** 参数设置为 **false**。
- 如果您还没有创建 ConfigMap 文件，请使用以下 YAML 代码创建该文件：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: dynamic-plugins-rhdh
data:
  dynamic-plugins.yaml: |
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: './dynamic-plugins/dist/janus-idp-backstage-plugin-analytics-provider-segment'
        disabled: false
```

b. 将 **dynamicPluginsConfigMapName** 参数的值设置为 **Backstage** 自定义资源中的 ConfigMap 文件的名称：

```
# ...
spec:
  application:
    dynamicPluginsConfigMapName: dynamic-plugins-rhdh
# ...
```

2. 保存配置更改。

5.3. 自定义遥测分割源

analytics-provider-segment 插件默认向红帽发送收集的遥测数据。但是，您可以配置一个新的 Segment 源，以根据您的需要接收遥测数据。对于配置，您需要一个指向分段源的唯一分段写键。

注意

通过配置新的分段源，您可以收集和分析 Telemetry 数据收集 部分中提到的相同数据集。您可能还需要为应用用户创建自己的遥测数据收集通知。

流程

1. 使用以下选项之一配置与分段源的集成：

使用 Helm Chart

- 在 Helm 配置文件中添加以下 YAML 代码：

```
# ...
upstream:
  backstage:
    extraEnvVars:
      - name: SEGMENT_WRITE_KEY
        value: <segment_key> 1
# ...
```

1 **<segment_key** > 表示 Segment 源的唯一标识符。

使用 Operator

- 在 **Backstage** 自定义资源(CR)中添加以下 YAML 代码：

```
# ...
extraEnvs:
  envs:
    - name: SEGMENT_WRITE_KEY
      value: <segment_key> 1
# ...
```

1 **<segment_key** > 表示 Segment 源的唯一标识符。

2. 保存配置更改。

第 6 章 在 OPENSHIFT CONTAINER PLATFORM 上为 RED HAT DEVELOPER HUB 启用可观察性

在 OpenShift Container Platform 中，指标通过 **/metrics** 规范名称下的 HTTP 服务端点公开。您可以创建一个 **ServiceMonitor** 自定义资源(CR)，从用户定义的项目中的服务端点提取指标。

6.1. 在 OPENSHIFT CONTAINER PLATFORM 集群的 HELM CHART 安装中启用指标监控

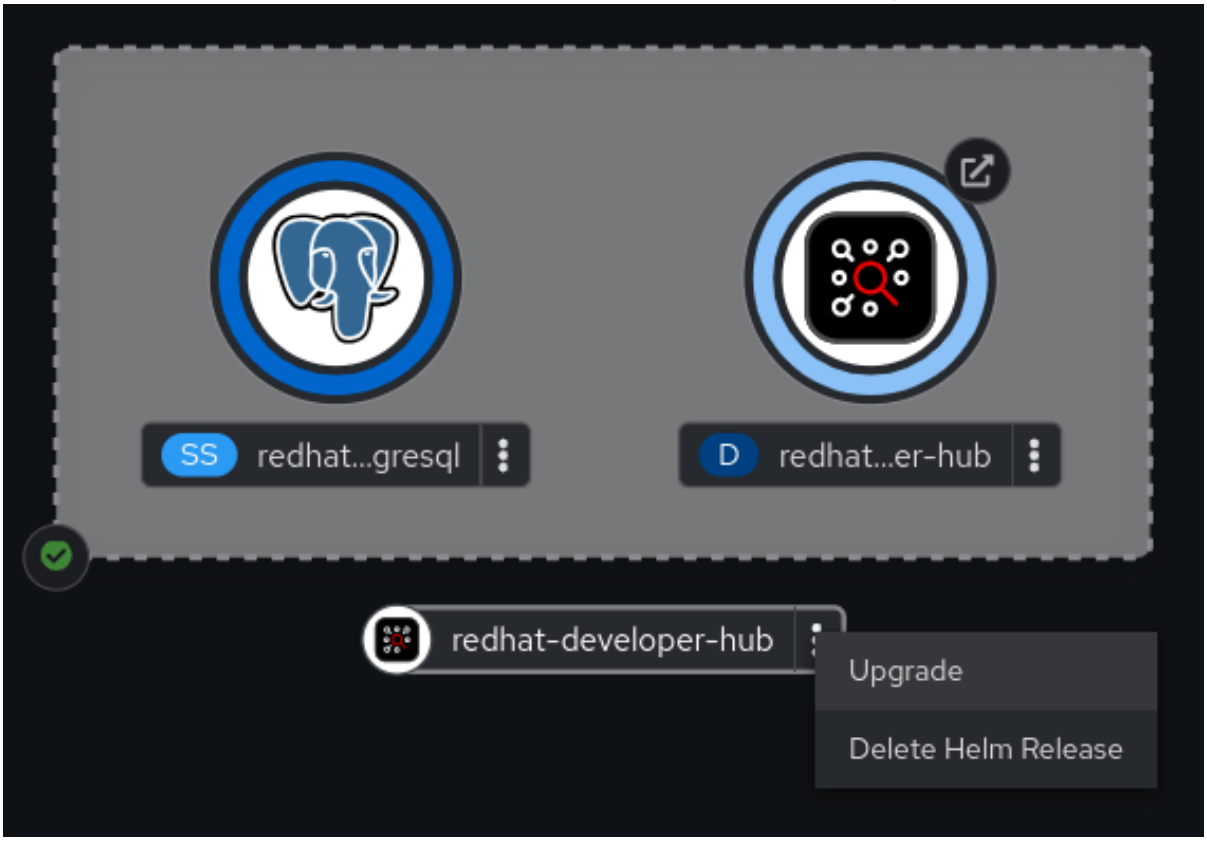
您可以从 OpenShift Container Platform Web 控制台的 Developer 视角为 Red Hat **Developer** Hub Helm 部署启用和查看指标。

先决条件

- 您的 OpenShift Container Platform 集群启用了 用户定义的项目的监控。
- 已使用 Helm Chart 在 OpenShift Container Platform 上安装 Red Hat Developer Hub。

流程

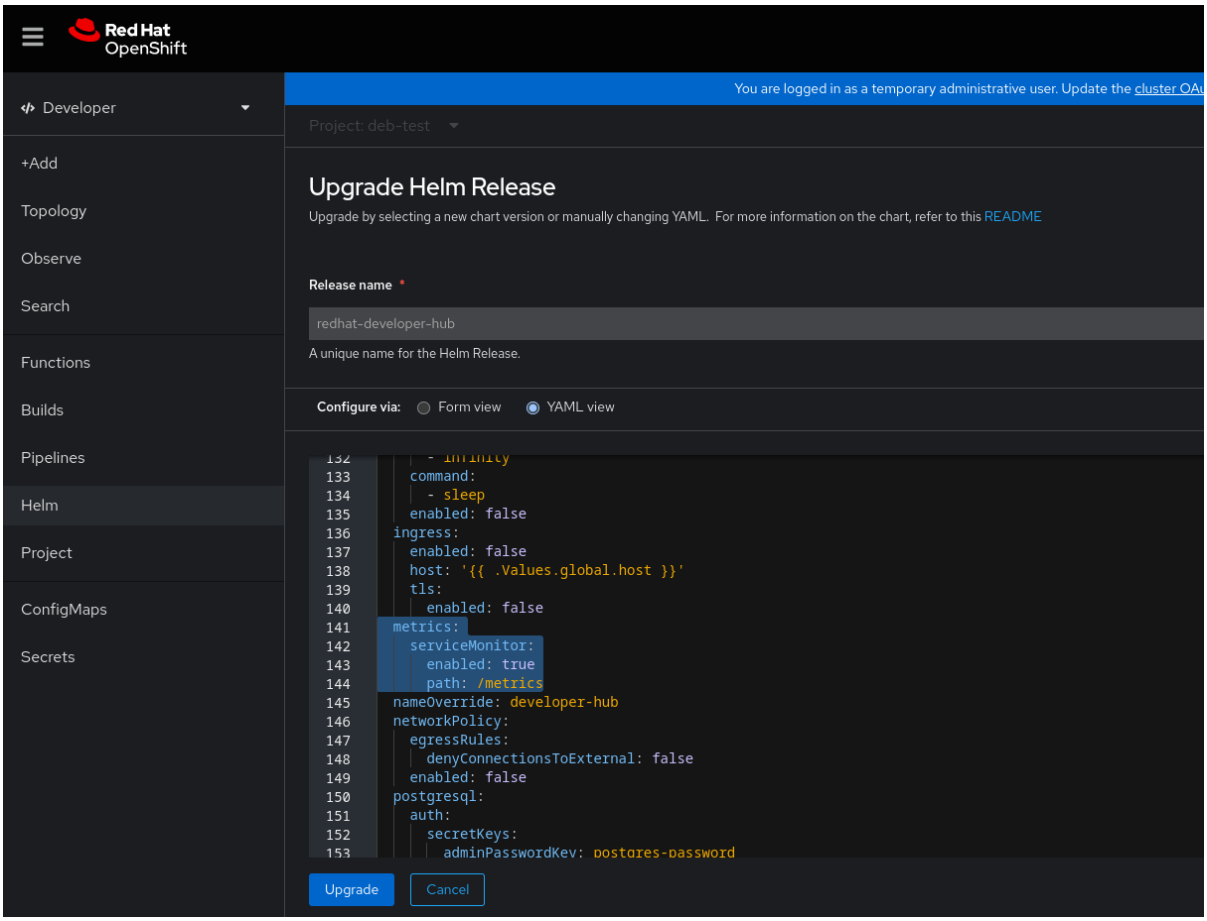
1. 从 OpenShift Container Platform Web 控制台中的 **Developer** 视角，选择 **Topology** 视图。
2. 点 Red Hat Developer Hub Helm Chart 的 overflow 菜单，然后选择 **Upgrade**。



3. 在 **Upgrade Helm Release** 页面中，选择 **Configure via** 中的 **YAML view** 选项，然后在 YAML 中配置 **metrics** 部分，如下例所示：

```
upstream:
# ...
  metrics:
```

```
    serviceMonitor:
      enabled: true
      path: /metrics
# ...
```



4. 单击 **Upgrade**。

验证

1. 从 OpenShift Container Platform web 控制台中的 **Developer** 视角，选择 **Observe** 视图。
2. 点 **Metrics** 选项卡查看 Red Hat Developer Hub pod 的指标。

6.2. 在 OPENSHIFT CONTAINER PLATFORM 集群上安装 RED HAT DEVELOPER HUB OPERATOR 中启用指标监控

您可以从 OpenShift Container Platform Web 控制台的 Developer 视角为 Operator 安装的 Red Hat **Developer** Hub 实例启用和查看指标。

先决条件

- 您的 OpenShift Container Platform 集群启用了 用户定义的项目的监控。
- 已使用 Red Hat Developer Hub Operator 在 OpenShift Container Platform 上安装 Red Hat Developer Hub。
- 已安装 OpenShift CLI(**oc**)。

流程

目前，Red Hat Developer Hub Operator 默认不支持创建 **ServiceMonitor** 自定义资源(CR)。您必须执行以下步骤来创建 **ServiceMonitor** CR，以从端点提取指标。

1. 将 **ServiceMonitor** CR 创建为 YAML 文件：

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: <custom_resource_name> 1
  namespace: <project_name> 2
  labels:
    app.kubernetes.io/instance: <custom_resource_name>
    app.kubernetes.io/name: backstage
spec:
  namespaceSelector:
    matchNames:
      - <project_name>
  selector:
    matchLabels:
      rhdh.redhat.com/app: backstage-<custom_resource_name>
  endpoints:
  - port: backend
    path: '/metrics'
```

1. 将 **<custom_resource_name** > 替换为 Red Hat Developer Hub CR 的名称。
2. 将 **<project_name** > 替换为运行 Red Hat Developer Hub 实例的 OpenShift Container Platform 项目的名称。

2. 运行以下命令来应用 **ServiceMonitor** CR：

```
oc apply -f <filename>
```

验证

1. 从 OpenShift Container Platform web 控制台中的 **Developer** 视角，选择 **Observe** 视图。
2. 点 **Metrics** 选项卡查看 Red Hat Developer Hub pod 的指标。

6.3. 其他资源

- OpenShift Container Platform - 管理指标

第 7 章 在公司代理后面运行 RHDH 应用程序

您可以在企业代理后面运行 RHDH 应用程序，方法是在启动应用程序前设置以下环境变量：

- **HTTP_PROXY** ：注意用于 HTTP 请求的代理。
- **https_PROXY** ：注意用于 HTTPS 请求的代理。

另外，您可以设置 **NO_PROXY** 环境变量，以便从代理中排除某些域。变量值是一个以逗号分隔的主机名列表，它们不需要代理才能被访问，即使指定了一个主机名。

7.1. 在 HELM 部署中配置代理信息

对于基于 Helm 的部署，开发者或集群管理员可在集群中创建资源，可以在 **values.yaml** Helm 配置文件中配置代理变量。

先决条件

- 已安装 Red Hat Developer Hub 应用程序。

流程

1. 在 Helm 配置文件中设置代理信息：

```
upstream:
  backstage:
    extraEnvVars:
      - name: HTTP_PROXY
        value: '<http_proxy_url>'
      - name: HTTPS_PROXY
        value: '<https_proxy_url>'
      - name: NO_PROXY
        value: '<no_proxy_settings>'
```

其中,

<http_proxy_url>

表示必须替换为 HTTP 代理 URL 的变量。

<https_proxy_url>

表示必须替换为 HTTPS 代理 URL 的变量。

<no_proxy_settings>

表示必须使用逗号分隔的 URL 替换的变量，该 URL 想从代理中排除，例如 **foo.com,baz.com**。

示例：使用 Helm Chart 设置代理变量

```
upstream:
  backstage:
    extraEnvVars:
      - name: HTTP_PROXY
        value: 'http://10.10.10.105:3128'
      - name: HTTPS_PROXY
```

```
      value: 'http://10.10.10.106:3128'
    - name: NO_PROXY
      value: 'localhost,example.org'
```

2. 保存配置更改。

7.2. 在 OPERATOR 部署中配置代理信息

对于基于 Operator 的部署，用于代理配置的方法取决于您的角色：

- 作为有权访问 Operator 命名空间的集群管理员，您可以在 Operator 的默认 ConfigMap 文件中配置代理变量。此配置将代理设置应用到 Operator 的所有用户。
- 作为开发者，您可以在自定义资源(CR)文件中配置代理变量。此配置将代理设置应用到从该 CR 创建的 RHDH 应用程序。

先决条件

- 已安装 Red Hat Developer Hub 应用程序。

流程

1. 根据您的角色执行以下步骤：
   - 作为管理员，在 Operator 的默认 ConfigMap 文件中设置代理信息：
     a. 在 default 命名空间 **rhdh-operator** 中搜索名为 **backstage-default-config** 的 ConfigMap 文件，并打开它。
     b. 查找 **deployment.yaml** 密钥。
     c. 在 **Deployment** spec 中设置 **HTTP_PROXY**、**HTTPS_PROXY** 和 **NO_PROXY** 环境变量的值，如下例所示：

示例：在 ConfigMap 文件中设置代理变量

```
# Other fields omitted
  deployment.yaml: |-
    apiVersion: apps/v1
    kind: Deployment
    spec:
      template:
        spec:
          # Other fields omitted
          initContainers:
            - name: install-dynamic-plugins
              # command omitted
              env:
                - name: NPM_CONFIG_USERCONFIG
                  value: /opt/app-root/src/.npmrc.dynamic-plugins
                - name: HTTP_PROXY
                  value: 'http://10.10.10.105:3128'
                - name: HTTPS_PROXY
                  value: 'http://10.10.10.106:3128'
```

```
        - name: NO_PROXY
          value: 'localhost,example.org'
      # Other fields omitted
  containers:
    - name: backstage-backend
      # Other fields omitted
      env:
        - name: APP_CONFIG_backend_listen_port
          value: "7007"
        - name: HTTP_PROXY
          value: 'http://10.10.10.105:3128'
        - name: HTTPS_PROXY
          value: 'http://10.10.10.106:3128'
        - name: NO_PROXY
          value: 'localhost,example.org'
```

- 作为开发者，在自定义资源(CR)文件中设置代理信息，如下例所示：

  示例：在 CR 文件中设置代理变量

```
spec:
  # Other fields omitted
  application:
    extraEnvs:
      envs:
        - name: HTTP_PROXY
          value: 'http://10.10.10.105:3128'
        - name: HTTPS_PROXY
          value: 'http://10.10.10.106:3128'
        - name: NO_PROXY
          value: 'localhost,example.org'
```

2. 保存配置更改。

第 8 章 RED HAT DEVELOPER HUB 与 AMAZON WEB SERVICES (AWS)集成

您可以将 Red Hat Developer Hub 应用程序与 Amazon Web Services (AWS)集成，这有助于简化 AWS 生态系统中的工作流。将 Developer Hub 资源与 AWS 集成，可访问全面的工具、服务和解决方案。

与 AWS 集成需要使用以下方法之一在 Elastic Kubernetes Service (EKS)中部署 Developer Hub：

- Helm chart
- Red Hat Developer Hub Operator

8.1. 使用 HELM CHART 在 ELASTIC KUBERNETES SERVICE (EKS)上部署 RED HAT DEVELOPER HUB

当您使用 Helm Chart 在 Elastic Kubernetes Service (EKS)中部署 Developer Hub 时，它会在 AWS 生态系统中编排强大的开发环境。

先决条件

- 已安装 AWS Application Load Balancer (ALB)附加组件的 EKS 集群。如需更多信息，请参阅 Amazon Developer Hub 上的应用程序负载均衡 和 安装 AWS Load Balancer Controller 附加组件。
- 已安装 **kubectl** CLI。
- 使用 **kubectl** 登录集群，并具有 **开发人员或 admin** 权限。
- 您已为 Developer Hub 实例配置了域名。域名可以是 Route 53 上的托管区条目，也可以是 AWS 外部管理的托管区条目。如需更多信息，请参阅配置 Amazon Route 53 作为您的 DNS 服务 文档。
- 在 AWS 证书管理器(ACM)中有一个您的首选域名的条目。确保保留您的证书 ARN 的记录。
- 您已订阅了 **registry.redhat.io**。如需更多信息，请参阅 Red Hat Container Registry 身份验证 。
- 您已在当前 **kubeconfig** 中将上下文设置为 EKS 集群。如需更多信息，请参阅为 Amazon EKS 集群创建或更新 kubeconfig 文件。
- 已安装 Helm 3 或最新的。如需更多信息，请参阅在 Amazon EKS 中使用 Helm。

流程

1. 进入终端，运行以下命令将包含 Developer Hub chart 的 Helm Chart 仓库添加到本地 Helm registry 中：

```
helm repo add openshift-helm-charts https://charts.openshift.io/
```

2. 创建并激活 < *rhdh*> 命名空间：

```
DEPLOYMENT_NAME=<redhat-developer-hub>
NAMESPACE=<rhdh>
kubectl create namespace ${NAMESPACE}
kubectl config set-context --current --namespace=${NAMESPACE}
```

3. 运行以下命令，创建一个 pull secret，用于从红帽生态系统拉取 Developer Hub 镜像：

```
kubectl create secret docker-registry rhdh-pull-secret \
  --docker-server=registry.redhat.io \
  --docker-username=<user_name> \ 1
  --docker-password=<password> \ 2
  --docker-email=<email> 3
```

1. 在命令中输入您的用户名。
2. 在命令中输入您的密码。
3. 在命令中输入您的电子邮件地址。

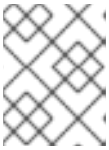
4. 使用以下模板创建一个名为 **values.yaml** 的文件：

```
global:
  # TODO: Set your application domain name.
  host: <your Developer Hub domain name>


route:
  enabled: false


upstream:
  service:
    # NodePort is required for the ALB to route to the Service
    type: NodePort


  ingress:
    enabled: true
    annotations:
      kubernetes.io/ingress.class: alb


      alb.ingress.kubernetes.io/scheme: internet-facing


      # TODO: Using an ALB HTTPS Listener requires a certificate for your own domain. Fill in the ARN of your certificate, e.g.:
      alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:xxx:xxxx:certificate/xxxxxx


      alb.ingress.kubernetes.io/listen-ports: '[{"HTTP": 80}, {"HTTPS":443}]'


      alb.ingress.kubernetes.io/ssl-redirect: '443'


      # TODO: Set your application domain name.
      external-dns.alpha.kubernetes.io/hostname: <your rhdh domain name>
```

```
backstage:
  image:
    pullSecrets:
    - rhdh-pull-secret
  podSecurityContext:
    # you can assign any random value as fsGroup
    fsGroup: 2000
postgresql:
  image:
    pullSecrets:
    - rhdh-pull-secret
  primary:
    podSecurityContext:
      enabled: true
      # you can assign any random value as fsGroup
      fsGroup: 3000
volumePermissions:
  enabled: true
```

5. 在终端中运行以下命令，以使用最新版本的 Helm Chart 并使用上一步中创建的 **values.yaml** 文件部署 Developer Hub：

```
helm install rhdh \
  openshift-helm-charts/redhat-developer-hub \
  [--version 1.2.1] \
  --values /path/to/values.yaml
```



注意

有关最新 chart 版本，请参阅 https://github.com/openshift-helm-charts/charts/tree/main/charts/redhat/redhat/redhat-developer-hub

6. 使用集群中的 Developer Hub 数据库密码和路由器基本 URL 值配置 Developer Hub Helm Chart 实例：

```
PASSWORD=$(kubectl get secret redhat-developer-hub-postgresql -o jsonpath="
{.data.password}" | base64 -d)
CLUSTER_ROUTER_BASE=$(kubectl get route console -n openshift-console -
o=jsonpath='{.spec.host}' | sed 's/^[^.]*\.//')
helm upgrade $DEPLOYMENT_NAME -i "https://github.com/openshift-helm-
charts/charts/releases/download/redhat-redhat-developer-hub-1.2.1/redhat-developer-hub-
1.2.1.tgz" \
   --set global.clusterRouterBase="$CLUSTER_ROUTER_BASE" \
   --set global.postgresql.auth.password="$PASSWORD"
```

7. 运行以下命令，显示正在运行的 Developer Hub 实例 URL：

```
echo "https://$DEPLOYMENT_NAME-$NAMESPACE.$CLUSTER_ROUTER_BASE"
```

验证

- 在浏览器中打开正在运行的 Developer Hub 实例 URL，以使用 Developer Hub。

Upgrade（升级）

Upgrade（升级）

- 要升级部署，请运行以下命令：

```
helm upgrade $DEPLOYMENT_NAME -i https://github.com/openshift-helm-charts/charts/releases/download/redhat-redhat-developer-hub-1.2.1/redhat-developer-hub-1.2.1.tgz
```

删除

- 要删除部署，请运行以下命令：

```
helm -n $NAMESPACE delete $DEPLOYMENT_NAME
```

8.2. 使用 OPERATOR 在 ELASTIC KUBERNETES SERVICE (EKS)上部署 RED HAT DEVELOPER HUB

您可以使用带有或不使用 Operator Lifecycle Manager (OLM)框架的 Red Hat Developer Hub Operator 在 EKS 上部署 Developer Hub。之后，您可以继续在 EKS 中安装 Developer Hub 实例。

8.2.1. 使用 OLM 框架安装 Red Hat Developer Hub Operator

先决条件

- 您已在当前 **kubeconfig** 中将上下文设置为 EKS 集群。如需更多信息，请参阅为 Amazon EKS 集群创建或更新 kubeconfig 文件。
- 已安装 **kubectl**。如需更多信息，请参阅 安装或更新 kubectl。
- 您已订阅了 **registry.redhat.io**。如需更多信息，请参阅 Red Hat Container Registry 身份验证 。
- 已安装 Operator Lifecycle Manager (OLM)。有关安装和故障排除的更多信息，请参阅 如何获取 Operator Lifecycle Manager？

流程

1. 在终端中运行以下命令，以创建安装 Operator 的 **rhdh-operator** 命名空间：

```
kubectl create namespace rhdh-operator
```

2. 使用以下命令创建 pull secret：

```
kubectl -n rhdh-operator create secret docker-registry rhdh-pull-secret \
    --docker-server=registry.redhat.io \
    --docker-username=<user_name> \ 1
    --docker-password=<password> \ 2
    --docker-email=<email> 3
```

1. 在命令中输入您的用户名。
2. 在命令中输入您的密码。
3. 在命令中输入您的电子邮件地址。

创建的 pull secret 用于从红帽生态系统中拉取 Developer Hub 镜像。

3. 创建一个 **CatalogSource** 资源，其中包含来自红帽生态系统的 Operator：

```
cat <<EOF | kubectl -n rhdh-operator apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: redhat-catalog
spec:
  sourceType: grpc
  image: registry.redhat.io/redhat/redhat-operator-index:v4.15
  secrets:
  - "rhdh-pull-secret"
  displayName: Red Hat Operators
EOF
```

4. 按如下方式创建 **OperatorGroup** 资源：

```
cat <<EOF | kubectl apply -n rhdh-operator -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: rhdh-operator-group
EOF
```

5. 使用以下代码创建 **Subscription** 资源：

```
cat <<EOF | kubectl apply -n rhdh-operator -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: rhdh
  namespace: rhdh-operator
spec:
  channel: fast
  installPlanApproval: Automatic
  name: rhdh
  source: redhat-catalog
  sourceNamespace: rhdh-operator
  startingCSV: rhdh-operator.v1.2.0
EOF
```

6. 运行以下命令验证创建的 Operator 是否正在运行：

```
kubectl -n rhdh-operator get pods -w
```

如果 Operator pod 显示 **ImagePullBackOff** 状态，则您可能需要直接在 Operator 部署清单中拉取镜像。

提示

您可以在 **deployment.spec.template.spec.imagePullSecrets** 列表中包含所需的 secret 名称，并使用 **kubectl get deployment -n rhdh-operator** 命令验证部署名称：

```
kubectl -n rhdh-operator patch deployment \
    rhdh.fast --patch '{"spec":{"template":{"spec":{"imagePullSecrets":[{"name":"rhdh-pull-secret"}]}}}}' \
    --type=merge
```

7. 更新 Operator 的默认配置，以确保 Developer Hub 资源可以按照以下流程在 EKS 中正确启动：

   a. 使用以下命令，编辑 **rhdh-operator** 命名空间中的 **backstage-default-config** ConfigMap：

   ```
   kubectl -n rhdh-operator edit configmap backstage-default-config
   ```

   b. 找到 **db-statefulset.yaml** 字符串，并将 **fsGroup** 添加到其 **spec.template.spec.securityContext** 中，如下例所示：

   ```
     db-statefulset.yaml: |
       apiVersion: apps/v1
       kind: StatefulSet
   --- TRUNCATED ---
       spec:
       --- TRUNCATED ---
         restartPolicy: Always
         securityContext:
         # You can assign any random value as fsGroup
           fsGroup: 2000
         serviceAccount: default
         serviceAccountName: default
   --- TRUNCATED ---
   ```

   c. 找到 **deployment.yaml** 字符串，并将 **fsGroup** 添加到规格中，如下例所示：

   ```
     deployment.yaml: |
       apiVersion: apps/v1
       kind: Deployment
   --- TRUNCATED ---
       spec:
         securityContext:
           # You can assign any random value as fsGroup
           fsGroup: 3000
         automountServiceAccountToken: false
   --- TRUNCATED ---
   ```

   d. 找到 **service.yaml** 字符串，并将 类**型改为 NodePort**，如下所示：

   ```
     service.yaml: |
       apiVersion: v1
       kind: Service
       spec:
   ```

```
    # NodePort is required for the ALB to route to the Service
    type: NodePort
--- TRUNCATED ---
```

e. 保存并退出。
等待几分钟，直到更改自动应用到 Operator pod。

8.2.2. 在没有 OLM 框架的情况下安装 Red Hat Developer Hub Operator

先决条件

- 已安装以下命令：
  - **git**
  - **make**
  - **sed**

流程

1. 使用以下命令将 Operator 存储库克隆到本地机器中：

```
git clone --depth=1 https://github.com/janus-idp/operator.git rhdh-operator && cd rhdh-operator
```

2. 运行以下命令并生成部署清单：

```
make deployment-manifest
```

以上命令生成一个名为 **rhdh-operator-<VERSION>.yaml** 的文件，该文件会被手动更新。

3. 运行以下命令在生成的部署清单中应用替换：

```
sed -i "s/backstage-operator/rhdh-operator/g" rhdh-operator-*.yaml
sed -i "s/backstage-system/rhdh-operator/g" rhdh-operator-*.yaml
sed -i "s/backstage-controller-manager/rhdh-controller-manager/g" rhdh-operator-*.yaml
```

4. 在编辑器中打开生成的部署清单文件并执行以下步骤：

a. 找到 **db-statefulset.yaml** 字符串，并将 **fsGroup** 添加到其 **spec.template.spec.securityContext** 中，如下例所示：

```
  db-statefulset.yaml: |
   apiVersion: apps/v1
   kind: StatefulSet
--- TRUNCATED ---
   spec:
   --- TRUNCATED ---
     restartPolicy: Always
     securityContext:
       # You can assign any random value as fsGroup
       fsGroup: 2000
```

```
      serviceAccount: default
      serviceAccountName: default
--- TRUNCATED ---
```

b. 找到 **deployment.yaml** 字符串，并将 **fsGroup** 添加到规格中，如下例所示：

```
  deployment.yaml: |
    apiVersion: apps/v1
    kind: Deployment
--- TRUNCATED ---
    spec:
      securityContext:
        # You can assign any random value as fsGroup
        fsGroup: 3000
      automountServiceAccountToken: false
--- TRUNCATED ---
```

c. 找到 **service.yaml** 字符串，并将 类**型改为 NodePort**，如下所示：

```
  service.yaml: |
    apiVersion: v1
    kind: Service
    spec:
      # NodePort is required for the ALB to route to the Service
      type: NodePort
--- TRUNCATED ---
```

d. 将默认镜像替换为红帽生态系统中拉取的镜像：

```
sed -i "s#gcr.io/kubebuilder/kube-rbac-proxy:.*#registry.redhat.io/openshift4/ose-kube-rbac-proxy:v4.15#g" rhdh-operator-*.yaml

sed -i "s#quay.io/janus-idp/operator:.*#registry.redhat.io/rhdh/rhdh-rhel9-operator:1.1#g" rhdh-operator-*.yaml

sed -i "s#quay.io/janus-idp/backstage-showcase:.*#registry.redhat.io/rhdh/rhdh-hub-rhel9:1.1#g" rhdh-operator-*.yaml

sed -i "s#quay.io/fedora/postgresql-15:.*#registry.redhat.io/rhel9/postgresql-15:latest#g" rhdh-operator-*.yaml
```

5. 将镜像 pull secret 添加到 Deployment 资源中的清单中，如下所示：

```
--- TRUNCATED ---

apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app.kubernetes.io/component: manager
    app.kubernetes.io/created-by: rhdh-operator
    app.kubernetes.io/instance: controller-manager
    app.kubernetes.io/managed-by: kustomize
    app.kubernetes.io/name: deployment
```

```
    app.kubernetes.io/part-of: rhdh-operator
    control-plane: controller-manager
  name: rhdh-controller-manager
  namespace: rhdh-operator
spec:
  replicas: 1
  selector:
    matchLabels:
      control-plane: controller-manager
  template:
    metadata:
      annotations:
        kubectl.kubernetes.io/default-container: manager
      labels:
        control-plane: controller-manager
    spec:
      imagePullSecrets:
        - name: rhdh-pull-secret
--- TRUNCATED ---
```

6. 使用以下命令应用清单来部署 Operator：

```
kubectl apply -f rhdh-operator-VERSION.yaml
```

7. 运行以下命令验证 Operator 是否正在运行：

```
kubectl -n rhdh-operator get pods -w
```

8.2.3. 在 EKS 中安装 Developer Hub 实例

安装并运行 Red Hat Developer Hub Operator 后，您可以在 EKS 中创建 Developer Hub 实例。

先决条件

- 已安装 AWS Application Load Balancer (ALB)附加组件的 EKS 集群。如需更多信息，请参阅 Amazon Elastic Kubernetes Service 上的应用程序负载均衡 和 安装 AWS Load Balancer Controller 附加组件。
- 您已为 Developer Hub 实例配置了域名。域名可以是 Route 53 上的托管区条目，也可以是 AWS 外部管理的托管区条目。如需更多信息，请参阅配置 Amazon Route 53 作为您的 DNS 服务 文档。
- 在 AWS 证书管理器(ACM)中有一个您的首选域名的条目。确保保留您的证书 ARN 的记录。
- 您已订阅了 **registry.redhat.io**。如需更多信息，请参阅 Red Hat Container Registry 身份验证 。
- 您已在当前 **kubeconfig** 中将上下文设置为 EKS 集群。如需更多信息，请参阅为 Amazon {eks} 集群创建或更新 kubeconfig 文件。
- 已安装 **kubectl**。如需更多信息，请参阅 安装或更新 kubectl。

流程

1. 使用以下模板，创建一个名为 **app-config-rhdh** 的 ConfigMap，其中包含 Developer Hub 配置：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config-rhdh
data:
  "app-config-rhdh.yaml": |
    app:
      title: Red Hat Developer Hub
      baseUrl: https://<rhdh_dns_name>
    backend:
      auth:
        keys:
          - secret: "${BACKEND_SECRET}"
      baseUrl: https://<rhdh_dns_name>
      cors:
        origin: https://<rhdh_dns_name>
```

2. 创建名为 **secrets-rhdh** 的 Secret，并添加名为 **BACKEND_SECRET** 的键，其值为 **Base64 编码的字符串** ：

```
apiVersion: v1
kind: Secret
metadata:
  name: secrets-rhdh
stringData:
  # TODO: See https://backstage.io/docs/auth/service-to-service-auth/#setup
  BACKEND_SECRET: "xxx"
```

重要

确保您为每个 Developer Hub 实例使用 **BACKEND_SECRET** 的唯一值。

您可以使用以下命令生成密钥：

```
node-p'require("crypto").randomBytes(24).toString("base64")'
```

3. 要启用从红帽生态系统目录中拉取 PostgreSQL 镜像，请在部署 Developer Hub 实例的命名空间中将镜像 pull secret 添加到 default 服务帐户中：

```
kubectl patch serviceaccount default \
   -p '{"imagePullSecrets": [{"name": "rhdh-pull-secret"}]}' \
   -n <your_namespace>
```

4. 使用以下模板创建自定义资源文件：

```
apiVersion: rhdh.redhat.com/v1alpha1
kind: Backstage
metadata:
 # TODO: this the name of your Developer Hub instance
  name: my-rhdh
spec:
  application:
    imagePullSecrets:
```

```
    - "rhdh-pull-secret"
  route:
    enabled: false
  appConfig:
    configMaps:
      - name: "app-config-rhdh"
  extraEnvs:
    secrets:
      - name: "secrets-rhdh"
```

5. 使用以下模板创建 Ingress 资源，确保根据需要自定义名称：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  # TODO: this the name of your Developer Hub Ingress
  name: my-rhdh
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing

    alb.ingress.kubernetes.io/target-type: ip

    # TODO: Using an ALB HTTPS Listener requires a certificate for your own domain. Fill in
the ARN of your certificate, e.g.:
    alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:us-xxx:xxxx:certificate/xxxxxx

     alb.ingress.kubernetes.io/listen-ports: '[{"HTTP": 80}, {"HTTPS":443}]'

    alb.ingress.kubernetes.io/ssl-redirect: '443'

    # TODO: Set your application domain name.
    external-dns.alpha.kubernetes.io/hostname: <rhdh_dns_name>

spec:
  ingressClassName: alb
  rules:
    # TODO: Set your application domain name.
    - host: <rhdh_dns_name>
      http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              # TODO: my-rhdh is the name of your Backstage Custom Resource.
              # Adjust if you changed it!
              name: backstage-my-rhdh
              port:
                name: http-backend
```

在前面的模板中，将 ' <rhdh_dns_name>' 替换为您的 Developer Hub 域名，并将 **alb.ingress.kubernetes.io/certificate-arn** 的值更新为您的证书 ARN。

验证

等待 DNS 名称响应，表示您的 Developer Hub 实例已准备就绪。

8.3. 在 RED HAT DEVELOPER HUB 中使用 AMAZON WEB SERVICES (AWS)监控和登录

在 Red Hat Developer Hub 中，通过 Amazon Web Services (AWS)集成促进监控和日志记录。借助 Amazon CloudWatch 用于实时监控和 Amazon Prometheus 等功能，您可以确保 AWS 基础架构上托管的 Developer Hub 应用程序的可靠性、可扩展性和合规性。

通过此集成，您可以监督、诊断和优化红帽生态系统中的应用程序，从而改进的开发和操作过程。

8.3.1. 使用 Amazon Prometheus 监控

Red Hat Developer Hub 提供与正在运行的应用程序相关的 Prometheus 指标。有关为 EKS 集群启用或部署 Prometheus 的更多信息，请参阅 Amazon 文档中的 Prometheus 指标。

要使用 Amazon Prometheus 监控 Developer Hub，您需要为 Prometheus 工作区创建一个 Amazon 受管服务，并配置 Developer Hub Prometheus 指标的 ingestion。如需更多信息，请参阅 Amazon 文档中的创建工作区 和 Ingest Prometheus metrics 部分。

将 Prometheus 指标放入创建的工作区后，您可以将指标提取配置为根据特定 pod 注解从 pod 中提取数据。

8.3.1.1. 为监控配置注解

您可以在 Helm 部署和 Operator 支持的部署中配置注解来监控。

Helm 部署

要注解用于监控的 backstage pod，请更新您的 **values.yaml** 文件，如下所示：

```
upstream:
  backstage:
    # --- TRUNCATED ---
    podAnnotations:
      prometheus.io/scrape: 'true'
      prometheus.io/path: '/metrics'
      prometheus.io/port: '7007'
      prometheus.io/scheme: 'http'
```

Operator 支持的部署

流程

1. 作为 Operator 的管理员，编辑默认配置以添加 Prometheus 注解，如下所示：

```
# Update OPERATOR_NS accordingly
OPERATOR_NS=rhdh-operator
kubectl edit configmap backstage-default-config -n "${OPERATOR_NS}"
```

2. 在 ConfigMap 中找到 **deployment.yaml** 键，并将注解添加到 **spec.template.metadata.annotations** 字段中，如下所示：

```
deployment.yaml: |-
  apiVersion: apps/v1
  kind: Deployment
```

```
# --- truncated ---
spec:
  template:
    # --- truncated ---
    metadata:
      labels:
        rhdh.redhat.com/app:  # placeholder for 'backstage-<cr-name>'
      # --- truncated ---
      annotations:
        prometheus.io/scrape: 'true'
        prometheus.io/path: '/metrics'
        prometheus.io/port: '7007'
        prometheus.io/scheme: 'http'
# --- truncated ---
```

3. 保存您的更改。

验证

验证提取是否正常工作：

1. 使用 **kubectl** 将 Prometheus 控制台转发到本地机器，如下所示：

```
kubectl --namespace=prometheus port-forward deploy/prometheus-server 9090
```

2. 打开 Web 浏览器，再进入到 **http://localhost:9090** 以访问 Prometheus 控制台。
3. 监控相关指标，如 **process_cpu_user_seconds_total**。

8.3.2. 使用 Amazon CloudWatch 日志进行日志记录

Red Hat Developer Hub 中的日志记录依赖于 winston 库。默认情况下，debug 级别的日志不会被记录。要激活调试日志，您必须在 Red Hat Developer Hub 实例中将环境变量 **LOG_LEVEL** 设置为 debug。

8.3.2.1. 配置应用程序日志级别

您可以在 Helm 部署和 Operator 支持的部署中配置应用程序日志级别。

Helm 部署

要更新日志记录级别，请将环境变量 **LOG_LEVEL** 添加到 Helm Chart 的 **values.yaml** 文件中：

```
upstream:
  backstage:
    # --- Truncated ---
    extraEnvVars:
      - name: LOG_LEVEL
        value: debug
```

Operator 支持的部署

您可以通过在自定义资源中包含环境变量 **LOG_LEVEL** 来修改日志记录级别，如下所示：

```
spec:
  # Other fields omitted
```

```
application:
  extraEnvs:
    envs:
      - name: LOG_LEVEL
        value: debug
```

8.3.2.2. 从 Amazon CloudWatch 检索日志

CloudWatch Container Insights 用于捕获 Amazon EKS 的日志和指标。如需更多信息，请参阅 Amazon EKS 文档的日志记录。

要捕获日志和指标，请 在集群中安装 Amazon CloudWatch Observability EKS 附加组件 。在 Container Insights 设置后，您可以使用 Logs Insights 或 Live Tail 视图访问容器日志。

CloudWatch 使用以下方法命名日志组，其中整合了所有容器日志：

/aws/containerinsights/<ClusterName>/application

以下是从 Developer Hub 实例检索日志的示例查询：

```
fields @timestamp, @message, kubernetes.container_name
| filter kubernetes.container_name in ["install-dynamic-plugins", "backstage-backend"]
```

8.4. 在 RED HAT DEVELOPER HUB 中使用 AMAZON COGNITO 作为身份验证供应商

在本节中，Amazon Cognito 是一个 AWS 服务，用于在 Developer Hub 中添加身份验证层。您可以使用用户池或通过第三方身份提供程序直接登录到 Developer Hub。

虽然 Amazon Cognito 不是 Developer Hub 的核心身份验证提供程序的一部分，但可以使用通用 OpenID Connect (OIDC)供应商集成。

您可以在 Helm Chart 和 Operator 支持的部署中配置 Developer Hub。

先决条件

- 您有一个 User Pool，或者您已创建一个新池。有关用户池的更多信息，请参阅 Amazon Cognito 用户池 文档。

注意

确保您已记下用户池所在的 AWS 区域，以及用户池 ID。

- 您已在用户池中创建了 App Client 来集成托管 UI。如需更多信息，请参阅使用 Amazon Cognito 控制台设置托管 UI。
  当使用 Amazon Cognito 控制台设置托管 UI 时，请确保进行以下调整：

  1. 在 **Allowed callback URL (s)** 部分中，包含 URL **https://<rhdh_url>/api/auth/oidc/handler/frame**。确保将 &lt **;rhdh_url** > 替换为您的 Developer Hub 应用程序的 URL，如 **my.rhdh.example.com**。

  2. 同样，在 **Allowed sign-out URL (s)** 部分中，添加 **https://<rhdh_url>**。将 **<rhdh_url** > 替换为您的 Developer Hub 应用程序的 URL，如 **my.rhdh.example.com**。

3. 在 **OAuth 2.0 授权类型** 下，选择 **Authorization code grant** 以返回授权代码。
4. 在 **OpenID Connect 范围** 下，确保至少选择以下范围：
   - OpenID
   - profile
   - 电子邮件

Helm 部署

流程

1. 编辑或创建自定义 **app-config-rhdh** ConfigMap，如下所示：

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

    signInPage: oidc
    auth:
      environment: production
      session:
        secret: ${AUTH_SESSION_SECRET}
      providers:
        oidc:
          production:
            clientId: ${AWS_COGNITO_APP_CLIENT_ID}
            clientSecret: ${AWS_COGNITO_APP_CLIENT_SECRET}
            metadataUrl: ${AWS_COGNITO_APP_METADATA_URL}
            callbackUrl: ${AWS_COGNITO_APP_CALLBACK_URL}
            scope: 'openid profile email'
            prompt: auto
```

2. 使用以下模板编辑或创建自定义 **secret-rhdh** Secret：

```
apiVersion: v1
kind: Secret
metadata:
  name: secrets-rhdh
stringData:
  AUTH_SESSION_SECRET: "my super auth session secret - change me!!!"
  AWS_COGNITO_APP_CLIENT_ID: "my-aws-cognito-app-client-id"
  AWS_COGNITO_APP_CLIENT_SECRET: "my-aws-cognito-app-client-secret"
  AWS_COGNITO_APP_METADATA_URL: "https://cognito-idp.
[region].amazonaws.com/[userPoolId]/.well-known/openid-configuration"
  AWS_COGNITO_APP_CALLBACK_URL:
"https://[rhdh_dns]/api/auth/oidc/handler/frame"
```

3. 在 **values.yaml** 文件中添加 ConfigMap 和 Secret 资源的引用：

```
upstream:
  backstage:
    image:
      pullSecrets:
      - rhdh-pull-secret
    podSecurityContext:
      fsGroup: 2000
    extraAppConfig:
      - filename: app-config-rhdh.yaml
        configMapRef: app-config-rhdh
    extraEnvVarsSecrets:
      - secrets-rhdh
```

4. 升级 Helm 部署：

```
helm upgrade rhdh \
  openshift-helm-charts/redhat-developer-hub \
  [--version 1.2.1] \
  --values /path/to/values.yaml
```

Operator 支持的部署

1. 将以下代码添加到 **app-config-rhdh** ConfigMap 中：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config-rhdh
data:
  "app-config-rhdh.yaml": |
    # --- Truncated ---

    signInPage: oidc
    auth:
      # Production to disable guest user login
      environment: production
      # Providing an auth.session.secret is needed because the oidc provider requires
session support.
      session:
        secret: ${AUTH_SESSION_SECRET}
      providers:
        oidc:
          production:
            # See https://github.com/backstage/backstage/blob/master/plugins/auth-
backend-module-oidc-provider/config.d.ts
            clientId: ${AWS_COGNITO_APP_CLIENT_ID}
            clientSecret: ${AWS_COGNITO_APP_CLIENT_SECRET}
            metadataUrl: ${AWS_COGNITO_APP_METADATA_URL}
            callbackUrl: ${AWS_COGNITO_APP_CALLBACK_URL}
            # Minimal set of scopes needed. Feel free to add more if needed.
            scope: 'openid profile email'

            # Note that by default, this provider will use the 'none' prompt which assumes
```

```
that your are already logged on in the IDP.
          # You should set prompt to:
          # - auto: will let the IDP decide if you need to log on or if you can skip login
when you have an active SSO session
          # - login: will force the IDP to always present a login form to the user
          prompt: auto
```

2. 将以下代码添加到 **secret-rhdh** Secret 中：

```
apiVersion: v1
kind: Secret
metadata:
  name: secrets-rhdh
stringData:
  # --- Truncated ---

  # TODO: Change auth session secret.
  AUTH_SESSION_SECRET: "my super auth session secret - change me!!!"

  # TODO: user pool app client ID
  AWS_COGNITO_APP_CLIENT_ID: "my-aws-cognito-app-client-id"

  # TODO: user pool app client Secret
  AWS_COGNITO_APP_CLIENT_SECRET: "my-aws-cognito-app-client-secret"

  # TODO: Replace region and user pool ID
  AWS_COGNITO_APP_METADATA_URL: "https://cognito-idp.
[region].amazonaws.com/[userPoolId]/.well-known/openid-configuration"

  # TODO: Replace <rhdh_dns>
  AWS_COGNITO_APP_CALLBACK_URL:
"https://[rhdh_dns]/api/auth/oidc/handler/frame"
```

3. 确保您的自定义资源包含对 **app-config-rhdh** ConfigMap 和 **secrets-rhdh** Secret 的引用：

```
apiVersion: rhdh.redhat.com/v1alpha1
kind: Backstage
metadata:
 # TODO: this the name of your Developer Hub instance
  name: my-rhdh
spec:
  application:
    imagePullSecrets:
    - "rhdh-pull-secret"
    route:
      enabled: false
    appConfig:
      configMaps:
        - name: "app-config-rhdh"
    extraEnvs:
      secrets:
        - name: "secrets-rhdh"
```

4. 可选：如果您有一个由自定义资源支持的现有 Developer Hub 实例，且没有编辑它，您可以手动删除 Developer Hub 部署以使用 Operator 重新创建它。运行以下命令以删除 Developer Hub 部署：

```
kubectl delete deployment -l app.kubernetes.io/instance=<CR_NAME>
```

验证

1. 进入 Developer Hub web URL 并使用 OIDC 身份验证签名，这会提示您通过配置的 AWS Cognito 用户池进行身份验证。
2. 登录后，访问 **Settings** 并验证用户详情。

第 9 章 RED HAT DEVELOPER HUB 与 MICROSOFT AZURE KUBERNETES SERVICE (AKS)集成

您可以将 Developer Hub 与 Microsoft Azure Kubernetes Service (AKS)集成，在开发中提供显著改进，为构建、部署和管理应用程序提供了一个简化的环境。

这个集成需要使用以下方法之一在 AKS 上部署 Developer Hub：

- Helm chart
- Red Hat Developer Hub Operator

9.1. 使用 HELM CHART 在 AZURE KUBERNETES SERVICE (AKS)上部署 RED HAT DEVELOPER HUB

您可以在 Azure Kubernetes Service (AKS)上部署 Developer Hub 应用程序，以访问用于构建、测试和部署应用程序的综合解决方案。

先决条件

- 您有一个带有有效订阅的 Azure 帐户。
- 已安装 Azure CLI。
- 已安装 **kubectl** CLI。
- 使用 **kubectl** 登录集群，并具有 **开发人员或 admin** 权限。
- 已安装 Helm 3 或最新的。

特定于 AKS 的基本 Developer Hub 部署的比较

- 权限问题 : Developer Hub 容器可能会遇到与 **权限** 相关的错误，如在尝试特定操作时 **Permission denied**。通过在 **PodSpec.securityContext** 中调整 **fsGroup** 来解决此错误。
- **Ingress 配置** ：在 AKS 中，配置入口对于访问已安装的 Developer Hub 实例至关重要。访问 Developer Hub 实例需要使用以下命令启用 Routing 附加组件（基于 NGINX 的 Ingress Controller）：

```
az aks approuting enable --resource-group <your_ResourceGroup> --name
<your_ClusterName>
```

提示

您可能需要安装 Azure CLI 扩展 **aks-preview**。如果没有自动安装扩展，您可能需要使用以下命令手动安装它：

```
az extension add --upgrade -n aks-preview --allow-preview true
```

注意

安装 Ingress Controller 后，集群中将部署带有 Ingress Controller 的 **app-routing-system** 命名空间。请注意已安装的 Ingress Controller 的 Developer Hub 应用程序地址（例如，108.141.70.228）），以便以后访问 Developer Hub 应用程序，稍后被引用为 < **app_address>**。

```
kubectl get svc nginx --namespace app-routing-system -o
jsonpath='{.status.loadBalancer.ingress[0].ip}'
```

- 命名空间 **管理** ：您可以使用以下命令为 AKS 中的 Developer Hub 部署创建一个专用命名空间：

```
kubectl create namespace <your_namespace>
```

流程

1. 运行以下命令登录到 AKS：

```
az login [--tenant=<optional_directory_name>]
```

2. 运行以下命令来创建资源组：

```
az group create --name <resource_group_name> --location <location>
```

提示

您可以运行以下命令来列出可用的区域：

```
az account list-locations -o table
```

3. 运行以下命令来创建 AKS 集群：

```
az aks create \
--resource-group <resource_group_name> \
--name <cluster_name> \
--enable-managed-identity \
--generate-ssh-keys
```

如需了解更多选项，请参阅 **--help**。

4. 运行以下命令来连接到集群：

```
az aks get-credentials --resource-group <resource_group_name> --name <cluster_name>
```

上一命令配置 Kubernetes 客户端，并将 **kubeconfig** 中的当前上下文设置为指向您的 AKS 集群。

5. 打开终端并运行以下命令来添加 Helm Chart 仓库：

```
helm repo add openshift-helm-charts https://charts.openshift.io/
```

6. 创建并激活 < *rhdh*> 命名空间：

```
DEPLOYMENT_NAME=<redhat-developer-hub>
NAMESPACE=<rhdh>
kubectl create namespace ${NAMESPACE}
kubectl config set-context --current --namespace=${NAMESPACE}
```

7. 运行以下命令，创建一个 pull secret，用于从红帽生态系统拉取 Developer Hub 镜像：

```
kubectl -n $NAMESPACE create secret docker-registry rhdh-pull-secret \
    --docker-server=registry.redhat.io \
    --docker-username=<redhat_user_name> \
    --docker-password=<redhat_password> \
    --docker-email=<email>
```

8. 使用以下模板创建一个名为 **values.yaml** 的文件：

```
global:
  host: <app_address>
route:
  enabled: false
upstream:
  ingress:
    enabled: true
    className: webapprouting.kubernetes.azure.com
    host:
  backstage:
    image:
      pullSecrets:
        - rhdh-pull-secret
    podSecurityContext:
      fsGroup: 3000
  postgresql:
    image:
      pullSecrets:
        - rhdh-pull-secret
    primary:
      podSecurityContext:
        enabled: true
        fsGroup: 3000
  volumePermissions:
    enabled: true
```

9. 要使用 Helm Chart 安装 Developer Hub，请运行以下命令：

```
helm -n $NAMESPACE install -f values.yaml $DEPLOYMENT_NAME openshift-helm-
charts/redhat-developer-hub --version 1.2.1
```

10. 验证部署状态：

```
kubectl get deploy $DEPLOYMENT_NAME -n $NAMESPACE
```

11. 使用集群中的 Developer Hub 数据库密码和路由器基本 URL 值配置 Developer Hub Helm Chart 实例：

```
PASSWORD=$(kubectl get secret redhat-developer-hub-postgresql -o jsonpath="
{.data.password}" | base64 -d)
CLUSTER_ROUTER_BASE=$(kubectl get route console -n openshift-console -
o=jsonpath='{.spec.host}' | sed 's/^[^.]*\.//')
helm upgrade $DEPLOYMENT_NAME -i "https://github.com/openshift-helm-
charts/charts/releases/download/redhat-redhat-developer-hub-1.2.1/redhat-developer-hub-
1.2.1.tgz" \
   --set global.clusterRouterBase="$CLUSTER_ROUTER_BASE" \
   --set global.postgresql.auth.password="$PASSWORD"
```

12. 运行以下命令，显示正在运行的 Developer Hub 实例 URL：

```
echo "https://$DEPLOYMENT_NAME-$NAMESPACE.$CLUSTER_ROUTER_BASE"
```

验证

- 在浏览器中打开正在运行的 Developer Hub 实例 URL，以使用 Developer Hub。

Upgrade（升级）

- 要升级部署，请运行以下命令：

```
helm upgrade $DEPLOYMENT_NAME -i https://github.com/openshift-helm-
charts/charts/releases/download/redhat-redhat-developer-hub-1.2.1/redhat-developer-hub-
1.2.1.tgz
```

删除

- 要删除部署，请运行以下命令：

```
helm -n $NAMESPACE delete $DEPLOYMENT_NAME
```

9.2. 使用 OPERATOR 在 AZURE KUBERNETES SERVICE (AKS)上部署 RED HAT DEVELOPER HUB

您可以使用 Red Hat Developer Hub Operator 在 AKS 上部署 Developer Hub。

先决条件

- 您有一个带有有效订阅的 Azure 帐户。
- 已安装 Azure CLI。
- 已安装 **kubectl** CLI。
- 使用 **kubectl** 登录集群，并具有 **开发人员或 admin** 权限。
- 已安装 Helm 3 或最新的。

特定于 AKS 的基本 Developer Hub 部署的比较

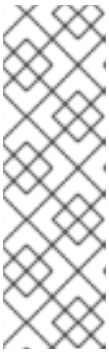
- 权限问题 : Developer Hub 容器可能会遇到与 **权限** 相关的错误，如在尝试特定操作时 **Permission denied**。通过在 **PodSpec.securityContext** 中调整 **fsGroup** 来解决此错误。
- **Ingress 配置** ：在 AKS 中，配置入口对于访问已安装的 Developer Hub 实例至关重要。访问 Developer Hub 实例需要使用以下命令启用 Routing 附加组件（基于 NGINX 的 Ingress Controller）：

```
az aks approuting enable --resource-group <your_ResourceGroup> --name
<your_ClusterName>
```

提示

您可能需要安装 Azure CLI 扩展 **aks-preview**。如果没有自动安装扩展，您可能需要使用以下命令手动安装它：

```
az extension add --upgrade -n aks-preview --allow-preview true
```



注意

安装 Ingress Controller 后，集群中将部署带有 Ingress Controller 的 **app-routing-system** 命名空间。请注意已安装的 Ingress Controller 的 Developer Hub 应用程序地址（例如，108.141.70.228）），以便以后访问 Developer Hub 应用程序，稍后被引用为 < **app_address>**。

```
kubectl get svc nginx --namespace app-routing-system -o
jsonpath='{.status.loadBalancer.ingress[0].ip}'
```

- 命名空间 **管理** ：您可以使用以下命令为 AKS 中的 Developer Hub 部署创建一个专用命名空间：

```
kubectl create namespace <your_namespace>
```

流程

1. 运行以下命令登录到 AKS：

```
az login [--tenant=<optional_directory_name>]
```

2. 运行以下命令来创建资源组：

```
az group create --name <resource_group_name> --location <location>
```

提示

您可以运行以下命令来列出可用的区域：

```
az account list-locations -o table
```

3. 运行以下命令来创建 AKS 集群：

```
az aks create \
```

```
--resource-group <resource_group_name> \
--name <cluster_name> \
--enable-managed-identity \
--generate-ssh-keys
```

如需了解更多选项，请参阅 **--help**。

4. 运行以下命令来连接到集群：

```
az aks get-credentials --resource-group <resource_group_name> --name <cluster_name>
```

上一命令配置 Kubernetes 客户端，并将 **kubeconfig** 中的当前上下文设置为指向您的 AKS 集群。

5. 获取名为 **rhdh-operator-<VERSION>.yaml** 的 Red Hat Developer Hub Operator 清单文件，并通过添加以下片段来修改 **db-statefulset.yaml** 和 **deployment.yaml** 的默认配置：

```
securityContext:
  fsGroup: 300
```

以下是清单中的指定位置：

```
db-statefulset.yaml: | spec.template.spec
deployment.yaml: | spec.template.spec
```

6. 将修改后的 Operator 清单应用到 Kubernetes 集群：

```
kubectl apply -f rhdh-operator-<VERSION>.yaml
```

注意

以上命令的执行是集群范围的，需要适当的集群权限。

7. 使用您的红帽凭证创建名为 **rhdh-pull-secret** 的 **ImagePull Secret**，以访问受保护的 **registry.redhat.io** 中的镜像，如下例所示：

```
kubectl -n <your_namespace> create secret docker-registry rhdh-pull-secret \
    --docker-server=registry.redhat.io \
    --docker-username=<redhat_user_name> \
    --docker-password=<redhat_password> \
    --docker-email=<email>
```

8. 创建名为 **rhdh-ingress.yaml** 的 Ingress 清单文件，指定您的 Developer Hub 服务名称，如下所示：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: rhdh-ingress
  namespace: <your_namespace>
spec:
  ingressClassName: webapprouting.kubernetes.azure.com
  rules:
```

```
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backstage-<your-CR-name>
                port:
                  name: http-backend
```

9. 要部署创建的 Ingress，请运行以下命令：

```
kubectl -n <your_namespace> apply -f rhdh-ingress.yaml
```

10. 使用以下示例，创建一个名为 **app-config-rhdh** 的 ConfigMap，其中包含 Developer Hub 配置：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config-rhdh
data:
  "app-config-rhdh.yaml": |
    app:
      title: Red Hat Developer Hub
      baseUrl: https://<app_address>
    backend:
      auth:
        keys:
          - secret: "${BACKEND_SECRET}"
      baseUrl: https://<app_address>
      cors:
        origin: https://<app_address>
```

11. 创建名为 **secrets-rhdh** 的 Secret，并添加一个名为 **BACKEND_SECRET** 的键，并带有 **Base64 编码的字符串** 值，如下例所示：

```
apiVersion: v1
kind: Secret
metadata:
  name: secrets-rhdh
stringData:
  BACKEND_SECRET: "xxx"
```

12. 创建名为 **rhdh.yaml** 的自定义资源(CR)清单文件，并包含之前创建的 **rhdh-pull-secret**，如下所示：

```
apiVersion: rhdh.redhat.com/v1alpha1
kind: Backstage
metadata:
  name: <your-rhdh-cr>
spec:
  application:
    imagePullSecrets:
```

```
    - rhdh-pull-secret
  appConfig:
    configMaps:
      - name: "app-config-rhdh"
  extraEnvs:
    secrets:
      - name: "secrets-rhdh"
```

13. 将 CR 清单应用到您的命名空间：

```
kubectl -n <your_namespace> apply -f rhdh.yaml
```

14. 使用 URL 访问部署的 Developer Hub：**https://<app_address&gt**;，其中 <app_address> 是之前获取的 Ingress 地址（例如 **https://108.141.70.228**）。

15. 可选：要删除 CR，请运行以下命令：

```
kubectl -n <your_namespace> delete -f rhdh.yaml
```

9.3. 在 RED HAT DEVELOPER HUB 中使用 AZURE KUBERNETES SERVICES (AKS)监控和日志记录

监控和日志记录是在 Red Hat Developer Hub 中管理和维护 Azure Kubernetes 服务(AKS)不可或缺的方面。借助管理 Prometheus Monitoring 和 Azure Monitor 集成等功能，管理员可以高效地监控资源利用率、诊断问题并确保容器化工作负载可靠性。

要启用 Managed Prometheus Monitoring，请使用 **az aks create** 或 **az aks update** 命令中的 **-enable-azure-monitor-metrics** 选项，具体取决于您是否创建新集群还是更新现有集群，例如：

```
az aks create/update --resource-group <your-ResourceGroup> --name <your-Cluster> --enable-azure-monitor-metrics
```

上一命令会安装指标附加组件，它会收集 Prometheus 指标。使用上一命令，您可以通过原生 Azure Monitor 指标和 Prometheus 指标启用对 Azure 资源的监控。您还可以在 **Monitoring → Insights** 下查看门户的结果。如需更多信息，请参阅使用 Azure Monitor 监控 Azure 资源。

另外，Managed Prometheus 服务和 Azure Monitor 的指标可以通过 Azure Managed Grafana 服务访问。如需更多信息，请参阅链接 Grafana 工作区 部分。

默认情况下，Prometheus 使用最小 ingesting 配置集，该配置集优化了 ingestion 卷，并为提取频率、目标和指标设置默认配置。可以通过自定义配置自定义默认设置。Azure 提供了各种方法，包括使用不同的 ConfigMap 来提供提取配置和其他指标附加组件设置。有关默认配置的更多信息，请参阅 Azure Monitor 中的默认 Prometheus 指标配置，并在 Azure Monitor managed service 中的 Customize extract of Prometheus metrics 部分。

9.3.1. 使用 Azure Kubernetes Services (AKS)查看日志

您可以访问 Kubernetes 对象生成的实时数据日志，并在 AKS 中的 Container Insights 中收集日志数据。

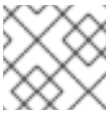
先决条件

- 您已在 AKS 上部署了 Developer Hub。如需更多信息，请参阅 第 9.1 节 “使用 Helm Chart 在 Azure Kubernetes Service (AKS)上部署 Red Hat Developer Hub”。

流程

查看 Developer Hub 实例的实时日志

1. 进入 Azure Portal。
2. 搜索资源组 < **your-ResourceGroup> 并找到您的** AKS 集群 < **your-Cluster>**。
3. 从菜单中选择 **Kubernetes resources → Workloads**。
4. 选择 & **lt;your-rhdh-cr>-developer-hub** （如果为 Helm Chart 安装），或 **<your-rhdh-cr>-backstage** （如果是 Operator 支持的安装）部署。
5. 单击左侧菜单中的 **Live Logs**。
6. 选择 pod。



注意

必须只有一个 pod。

实时日志数据会被收集并显示。

查看容器引擎中的实时日志数据

1. 进入 Azure Portal。
2. 搜索资源组 < **your-ResourceGroup> 并找到您的** AKS 集群 < **your-Cluster>**。
3. 从菜单中选择 **Monitoring → Insights**。
4. 转至" **容器**" 选项卡。
5. 找到 backend-backstage 容器，并点击它来查看 Container Engine 生成的实时日志数据。

9.4. 在 RED HAT DEVELOPER HUB 中使用 MICROSOFT AZURE 作为身份验证供应商

Developer Hub 中的 **core-plugin-api** 软件包与 Microsoft Azure 身份验证供应商集成，使用 Azure OAuth 进行身份验证。

先决条件

- 您已在 AKS 上部署了 Developer Hub。如需更多信息，请参阅 第 9.1 节 “使用 Helm Chart 在 Azure Kubernetes Service (AKS)上部署 Red Hat Developer Hub”。
- 您已在 Azure 门户中注册了应用程序。如需更多信息，请参阅使用 Microsoft 身份平台注册应用程序。

9.4.1. 在 Helm 部署中使用 Microsoft Azure 作为身份验证供应商

在使用 Helm Chart 安装时，您可以在 Red Hat Developer Hub 中使用 Microsoft Azure 作为身份验证供应商。如需更多信息，请参阅 第 9.1 节 “使用 Helm Chart 在 Azure Kubernetes Service (AKS)上部署 Red Hat Developer Hub”。

流程

1. 注册应用程序后，请注意以下几点：
   - **clientId**: Application (client) ID，它位于 App **Registration → Overview** 下。
   - **clientSecret**: Secret，在 *App Registration → Certificates & secrets 下找到（如果需要，创建新的）。
   - **tenantId**: Directory (tenant) ID，它位于 **App Registration → Overview** 下。
2. 确保 Developer Hub ConfigMap 中包含以下片段：

```
auth:
  environment: production
  providers:
    microsoft:
      production:
        clientId: ${AZURE_CLIENT_ID}
        clientSecret: ${AZURE_CLIENT_SECRET}
        tenantId: ${AZURE_TENANT_ID}
        domainHint: ${AZURE_TENANT_ID}
        additionalScopes:
          - Mail.Send
```

   您可以创建新文件或将其添加到现有文件中。

3. 将 ConfigMap 应用到 Kubernetes 集群：

```
kubectl -n <your_namespace> apply -f <app-config>.yaml
```

4. 创建或重复使用包含 Azure 凭证的现有 Secret，并添加以下片段：

```
stringData:
  AZURE_CLIENT_ID: <value-of-clientId>
  AZURE_CLIENT_SECRET: <value-of-clientSecret>
  AZURE_TENANT_ID: <value-of-tenantId>
```

5. 将 secret 应用到 Kubernetes 集群：

```
kubectl -n <your_namespace> apply -f <azure-secrets>.yaml
```

6. 确保您的 **values.yaml** 文件引用之前创建的 ConfigMap 和 Secret：

```
upstream:
  backstage:
  ...
    extraAppConfig:
      - filename: ...
        configMapRef: <app-config-containing-azure>
    extraEnvVarsSecrets:
      - <secret-containing-azure>
```

7. 可选： 如果已安装 Helm Chart，请升级它：

```
helm -n <your_namespace> upgrade -f <your-values.yaml> <your_deploy_name> redhat-developer/backstage --version 1.2.1
```

8. 可选： 如果没有更改 **rhdh.yaml** 文件，例如，您只更新从它引用的 ConfigMap 和 Secret，请通过删除对应的 pod 来刷新 Developer Hub 部署：

```
kubectl -n <your_namespace> delete pods -l backstage.io/app=backstage-<your-rhdh-cr>
```

9.4.2. 在 Operator 支持的部署中使用 Microsoft Azure 作为身份验证供应商

在使用 Operator 安装时，您可以在 Red Hat Developer Hub 中使用 Microsoft Azure 作为身份验证供应商。如需更多信息，请参阅 第 2.2 节 “使用 Operator 在 OpenShift Container Platform 上部署 Red Hat Developer Hub”。

流程

1. 注册应用程序后，请注意以下几点：
   - **clientId**: Application (client) ID，它位于 App **Registration → Overview** 下。
   - **clientSecret**: Secret，在 *App Registration → Certificates & secrets 下找到（如果需要，创建新的）。
   - **tenantId**: Directory (tenant) ID，它位于 **App Registration → Overview** 下。
2. 确保 Developer Hub ConfigMap 中包含以下片段：

```
auth:
  environment: production
  providers:
    microsoft:
      production:
        clientId: ${AZURE_CLIENT_ID}
        clientSecret: ${AZURE_CLIENT_SECRET}
        tenantId: ${AZURE_TENANT_ID}
        domainHint: ${AZURE_TENANT_ID}
        additionalScopes:
          - Mail.Send
```

   您可以创建新文件或将其添加到现有文件中。

3. 将 ConfigMap 应用到 Kubernetes 集群：

```
kubectl -n <your_namespace> apply -f <app-config>.yaml
```

4. 创建或重复使用包含 Azure 凭证的现有 Secret，并添加以下片段：

```
stringData:
  AZURE_CLIENT_ID: <value-of-clientId>
  AZURE_CLIENT_SECRET: <value-of-clientSecret>
  AZURE_TENANT_ID: <value-of-tenantId>
```

5. 将 secret 应用到 Kubernetes 集群：

```
kubectl -n <your_namespace> apply -f <azure-secrets>.yaml
```

6. 确保自定义资源清单包含对之前创建的 ConfigMap 和 Secret 的引用：

```
apiVersion: rhdh.redhat.com/v1alpha1
kind: Backstage
metadata:
  name: <your-rhdh-cr>
spec:
  application:
    imagePullSecrets:
    - rhdh-pull-secret
    route:
      enabled: false
    appConfig:
      configMaps:
        - name: <app-config-containing-azure>
    extraEnvs:
      secrets:
        - name: <secret-containing-azure>
```

7. 应用自定义资源清单：

```
kubectl -n <your_namespace> apply -f rhdh.yaml
```

8. 可选：如果没有更改 **rhdh.yaml** 文件，例如，您只更新从它引用的 ConfigMap 和 Secret，请通过删除对应的 pod 来刷新 Developer Hub 部署：

```
kubectl -n <your_namespace> delete pods -l backstage.io/app=backstage-<your-rhdh-cr>
```

第 10 章 RED HAT DEVELOPER HUB 中的基于角色的访问控制 (RBAC)

基于角色的访问控制是一种安全范例，限制对授权用户访问。此功能包括定义具有特定权限的角色，然后将这些角色分配给用户。

Red Hat Developer Hub 使用 RBAC 来提高平台中的权限系统。Developer Hub 中的 RBAC 功能引入了一个管理员角色，并通过促进高效的访问控制来利用组织结构，包括团队、组和用户。

10.1. RED HAT DEVELOPER HUB 中的权限策略

Red Hat Developer Hub 中的权限策略是一组规则，用于控制对资源或功能的访问。这些策略声明根据其角色授予用户的授权级别。实施权限策略，以维护给定环境中的安全性和保密性。

您可以在 Developer Hub 中定义以下类型的权限：

- 资源类型
- 基本的

两种权限类型之间的区别取决于权限是否包含定义的资源类型。

您可以使用关联的资源类型或权限名称来定义资源类型权限，如下例所示：

资源类型权限定义示例

```
p, role:default/myrole, catalog.entity.read, read, allow
g, user:default/myuser, role:default/myrole

p, role:default/another-role, catalog-entity, read, allow
g, user:default/another-user, role:default/another-role
```

您可以使用权限名称在 Developer Hub 中定义基本权限，如下例所示：

基本权限定义示例

```
p, role:default/myrole, catalog.entity.create, create, allow
g, user:default/myuser, role:default/myrole
```

Developer Hub 支持以下权限策略：

目录权限

Name	资源类型	policy	描述
catalog.entity.read	**catalog-entity**	读取	允许用户或角色从目录中读取
catalog.entity.create		create	允许用户或团队创建目录实体，包括在目录中注册现有组件

Name	资源类型	policy	描述
catalog.entity.refresh	**catalog-entity**	update	允许用户或组从目录中刷新一个或多个实体
catalog.entity.delete	**catalog-entity**	delete	允许用户或组从目录中删除一个或多个实体
catalog.location.read		读取	允许用户或组从目录中读取一个或多个位置
catalog.location.create		create	允许用户或组在目录中创建位置
catalog.location.delete		delete	允许用户或组从目录中删除位置

Scaffolder 权限

Name	资源类型	policy	描述
scaffolder.action.execute	**scaffolder-action**		允许从模板执行操作
scaffolder.template.parameter.read	**scaffolder-template**	读取	允许用户或组从模板读取一个或多个参数
scaffolder.template.step.read	**scaffolder-template**	读取	允许用户或角色从模板读取一个或多个步骤
scaffolder.task.create		create	允许用户或角色触发软件模板，以创建新的构建程序任务
scaffolder.task.cancel			允许用户或角色取消当前运行的构建程序任务
scaffolder.task.read		读取	允许用户或角色读取所有构建器任务及其关联的事件和日志

RBAC 权限

名称	资源类型	policy	描述
policy.entity.read	**policy-entity**	读取	允许用户或角色读取权限策略和角色
policy.entity.create	**policy-entity**	create	允许用户或组创建一个或多个权限策略和角色
policy.entity.update	**policy-entity**	update	允许用户或组更新一个或多个权限策略和角色
policy.entity.delete	**policy-entity**	delete	允许用户或组删除一个或多个权限策略和角色

Kubernetes 权限

名称	资源类型	policy	描述
kubernetes.proxy			允许用户或组访问代理端点

OCM 权限

名称	资源类型	policy	描述
ocm.entity.read		读取	允许从 OCM 插件读取用户或角色
ocm.cluster.read		读取	允许用户或角色读取 OCM 插件中的集群信息

拓扑权限

名称	资源类型	policy	描述
topology.view.read		读取	允许用户或角色查看拓扑插件
kubernetes.proxy			允许用户或角色访问代理端点，允许他们读取 RHDH 中的 pod 日志和事件

10.1.1. 权限策略配置

在 Red Hat Developer Hub 中配置权限策略有两种方法，包括：

- 配置权限策略管理员
- 配置外部文件中定义的权限策略

10.1.1.1. 配置权限策略管理员

Developer Hub 中用户和组的权限策略由权限策略管理员管理。只有权限策略管理员可以访问基于角色的访问控制 REST API。

配置策略管理员的目的是启用特定、受限制的经过身份验证的用户来访问 RBAC REST API。权限策略在 **policy.csv** 文件中定义，该文件在 **app-config-rhdh** ConfigMap 中引用。OpenShift 平台管理员或集群管理员可以执行此任务，可访问部署 Red Hat Developer Hub 的命名空间。

您可以在 **app-config.yaml** 文件中设置权限策略管理员的凭证，如下所示：

```
permission:
  enabled: true
  rbac:
    admin:
      users:
        - name: user:default/joeuser
```

权限策略角色(**role:default/rbac_admin**)在 Developer Hub 中是一个默认角色，在创建时包括一些权限，如创建、读取、更新和删除权限策略/角色，以及从目录中读取。

如果默认权限不适合您的要求，您可以使用相关权限策略根据您的要求定义新的管理员角色。另外，您可以使用可选的 **superUsers** 配置值，该值在 Developer Hub 间授予不受限制的权限。

您可以在 **app-config.yaml** 文件中设置 **superUsers**，如下所示：

```
# ...
permission:
  enabled: true
  rbac:
    admin:
      superUsers:
        - name: user:default/joeuser
        # ...
```

10.1.1.2. 配置外部文件中定义的权限策略

您可以在启动 Red Hat Developer Hub 前配置权限策略。如果外部文件中定义了权限策略，您可以在 Developer Hub 中导入同一文件。您必须使用以下 Casbin 规则格式定义权限策略：

```
---
`p, <ROLE>, <PERMISSION_NAME or PERMISSION_RESOURCE_TYPE>,
<PERMISSION_POLICY_ACTION>, <ALLOW or DENY>`
---
```

您可以使用以下 Casbin 规则格式定义角色：

```
---
`g, <USER or GROUP>, <ROLE>`
---
```

注意

有关 Casbin 规则格式的详情，请参考 Casbin 规则 的基本信息。

以下是权限策略配置示例：

```
---
`p, role:default/guests, catalog-entity, read, allow`
```

p, role:default/guests, catalog.entity.create, create, allow

g, user:default/<USER_TO_ROLE>, role:default/guests

g, group:default/<GROUP_TO_ROLE>, role:default/guests --

如果定义的权限不包含与其关联的操作，请添加 **use** 作为策略。请参见以下示例：

```
---
`p, role:default/guests, kubernetes.proxy, use, allow`
---
```

您可以在 **app-config.yaml** 文件中定义 **policy.csv** 文件路径：

```
permission:
  enabled: true
  rbac:
    policies-csv-file: /some/path/rbac-policy.csv
```

您可以使用可选配置值来启用重新载入 CSV 文件，而无需重启 Developer Hub 实例。

在 **app-config.yaml** 文件中设置 **policyFileReload** 选项的值：

```
# ...
permission:
  enabled: true
  rbac:
    policies-csv-file: /some/path/rbac-policy.csv
    policyFileReload: true
    # ...
```

10.1.1.2.1. 将 policy.csv 文件挂载到 Developer Hub Helm Chart 中

当使用 Helm Chart 部署 Red Hat Developer Hub 时，您必须通过将 **policy.csv** 文件挂载到 Developer Hub Helm Chart 来定义 policy.csv 文件。

您可以通过创建一个 **configMap** 并挂载它，将 **policy.csv** 文件添加到 Developer Hub Helm Chart 中。

先决条件

- 使用 OpenShift Container Platform Web 控制台登录到 OpenShift Container Platform 帐户。
- Red Hat Developer Hub 使用 Helm Chart 安装和部署。

有关使用 Helm Chart 在 OpenShift Container Platform 上安装 Red Hat Developer Hub 的更多信息，请参阅 第 2.1 节 “使用 Helm Chart 在 OpenShift Container Platform 上部署 Red Hat Developer Hub”。

流程

1. 在 OpenShift Container Platform 中，创建一个 ConfigMap 来保存策略，如下例所示：

   ConfigMap 示例

   ```
   kind: ConfigMap
   apiVersion: v1
   metadata:
     name: rbac-policy
     namespace: rhdh
   data:
     rbac-policy.csv: |
       p, role:default/guests, catalog-entity, read, allow
       p, role:default/guests, catalog.entity.create, create, allow

       g, user:default/<YOUR_USER>, role:default/guests
   ```

2. 在 Developer Hub Helm Chart 中，进入 **Root Schema → Backstage chart schema → Backstage parameters → Backstage container additional volume mount**。

3. 选择 **Add Backstage 容器附加卷挂载** 并添加以下值：

   - **mountPath:opt/app-root/src/rbac**
   - **名称 ：rbac-policy**

4. 将 RBAC 策略添加到 Developer Hub Helm Chart 中的 **Backstage 容器额外卷中** ：

   - **名称 ：rbac-policy**
   - configMap
     - **DefaultMode:420**
     - **名称 ：rbac-policy**

5. 更新 **app-config.yaml** 文件中的策略路径，如下所示：

   app-config.yaml 文件示例

   ```
   permission:
     enabled: true
     rbac:
       policies-csv-file: ./rbac/rbac-policy.csv
   ```

10.2. RED HAT DEVELOPER HUB 中的条件策略

Red Hat Developer Hub 中的权限框架提供了由 RBAC 后端插件支持的条件(**backstage-plugin-rbac-backend**)。条件作为 RBAC 后端插件提供的 Developer Hub 资源的内容过滤器。

RBAC 后端 API 存储分配给数据库中的角色的条件。当您请求访问 frontend 资源时，RBAC 后端 API 会搜索对应的条件，并使用其插件 ID 将它们委派给适当的插件。如果您分配到具有不同条件的多个角色，则 RBAC 后端将使用 **anyOf** 条件合并条件。

条件条件条件

Developer Hub 中的条件是带有规则和参数的简单条件。但是，条件也可以包含参数或按条件条件组合的参数数组。支持的条件条件包括：

- **allOf**: 确保阵列中的所有条件都必须为 true，才能满足组合条件。
- **anyOf**: 确保该阵列中至少需要满足组合条件的条件。
- Not : 确保其中的条件不能满足组合条件。

条件对象

该插件指定条件支持的参数。您可以从 RBAC API 端点访问条件对象模式，以了解如何构建条件 JSON 对象，然后由 RBAC 后端插件 API 使用该对象。

条件对象包含以下参数：

表 10.1. 条件对象参数

参数	类型	描述
result	字符串	始终值为 **CONDITIONAL**
roleEntityRef	字符串	对 RBAC 角色的字符串实体引用，如 **role:default/dev**
pluginId	字符串	对应的插件 ID，如 **目录**
permissionMapping	字符串数组	数组权限操作，如 **['read', 'update', 'delete']**
resourceType	字符串	插件提供的资源类型，如 **catalog-entity**
conditions	JSON	带有参数或数组参数的条件 JSON，按条件加入

10.2.1. 条件策略定义

您可以访问 Red Hat Developer Hub 中条件策略的 API 端点。例如，要检索可用的条件规则，这有助于定义这些策略，您可以访问 GET [api/plugins/condition-rules] 端点。

api/plugins/condition-rules 返回条件参数 schemas，例如：


```
[
  {
    "pluginId": "catalog",
    "rules": [
      {
        "name": "HAS_ANNOTATION",
        "description": "Allow entities with the specified annotation",
        "resourceType": "catalog-entity",
        "paramsSchema": {
          "type": "object",
          "properties": {
            "annotation": {
              "type": "string",
              "description": "Name of the annotation to match on"
            },
            "value": {
              "type": "string",
              "description": "Value of the annotation to match on"
            }
          },
          "required": [
            "annotation"
          ],
          "additionalProperties": false,
          "$schema": "http://json-schema.org/draft-07/schema#"
        }
      },
      {
        "name": "HAS_LABEL",
        "description": "Allow entities with the specified label",
        "resourceType": "catalog-entity",
        "paramsSchema": {
          "type": "object",
          "properties": {
            "label": {
              "type": "string",
              "description": "Name of the label to match on"
            }
          },
          "required": [
            "label"
          ],
          "additionalProperties": false,
          "$schema": "http://json-schema.org/draft-07/schema#"
        }
      },
      {
        "name": "HAS_METADATA",
```

```
    "description": "Allow entities with the specified metadata subfield",
    "resourceType": "catalog-entity",
    "paramsSchema": {
      "type": "object",
      "properties": {
        "key": {
          "type": "string",
          "description": "Property within the entities metadata to match on"
        },
        "value": {
          "type": "string",
          "description": "Value of the given property to match on"
        }
      },
      "required": [
        "key"
      ],
      "additionalProperties": false,
      "$schema": "http://json-schema.org/draft-07/schema#"
    }
  },
  {
    "name": "HAS_SPEC",
    "description": "Allow entities with the specified spec subfield",
    "resourceType": "catalog-entity",
    "paramsSchema": {
      "type": "object",
      "properties": {
        "key": {
          "type": "string",
          "description": "Property within the entities spec to match on"
        },
        "value": {
          "type": "string",
          "description": "Value of the given property to match on"
        }
      },
      "required": [
        "key"
      ],
      "additionalProperties": false,
      "$schema": "http://json-schema.org/draft-07/schema#"
    }
  },
  {
    "name": "IS_ENTITY_KIND",
    "description": "Allow entities matching a specified kind",
    "resourceType": "catalog-entity",
    "paramsSchema": {
      "type": "object",
      "properties": {
        "kinds": {
          "type": "array",
          "items": {
            "type": "string"
          },
```

```
            "description": "List of kinds to match at least one of"
          }
        },
        "required": [
          "kinds"
        ],
        "additionalProperties": false,
        "$schema": "http://json-schema.org/draft-07/schema#"
      }
    },
    {
      "name": "IS_ENTITY_OWNER",
      "description": "Allow entities owned by a specified claim",
      "resourceType": "catalog-entity",
      "paramsSchema": {
        "type": "object",
        "properties": {
          "claims": {
            "type": "array",
            "items": {
              "type": "string"
            },
            "description": "List of claims to match at least one on within ownedBy"
          }
        },
        "required": [
          "claims"
        ],
        "additionalProperties": false,
        "$schema": "http://json-schema.org/draft-07/schema#"
      }
    }
  ]
 }
 ... <another plugin condition parameter schemas>
]
```

RBAC 后端 API 根据前面的条件模式构造一个条件 JSON 对象。

10.2.1.1. 条件策略示例

在 Red Hat Developer Hub 中，您可以使用或没有条件定义条件策略。您可以使用以下示例根据您的用例定义条件：

没有条件的条件

只有用户是所有者组的成员时，请考虑没有条件显示目录的条件。要添加此条件，您可以使用目录插件模式 IS_ENTITY_OWNER，如下所示：

没有条件的示例

```
{
  "rule": "IS_ENTITY_OWNER",
  "resourceType": "catalog-entity",
  "params": {
    "claims": ["group:default/team-a"]
  }
}
```

在上例中，唯一使用的条件参数是 claims，其中包含用户或组实体引用的列表。

您可以通过添加额外的参数将前面的示例条件应用到 RBAC REST API，如下所示：

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/test",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["read"],
  "conditions": {
    "rule": "IS_ENTITY_OWNER",
    "resourceType": "catalog-entity",
    "params": {
      "claims": ["group:default/team-a"]
    }
  }
}
```

具有条件的条件

考虑条件条件，它只有在用户是所有者组成员或显示所有目录用户组的列表时才会显示目录。

要添加条件，您可以在条件中添加另一个规则作为 IS_ENTITY_KIND，如下所示：

带有条件的条件示例

```
{
  "anyOf": [
    {
      "rule": "IS_ENTITY_OWNER",
```

```
      "resourceType": "catalog-entity",
      "params": {
        "claims": ["group:default/team-a"]
      }
    },
    {
      "rule": "IS_ENTITY_KIND",
      "resourceType": "catalog-entity",
      "params": {
        "kinds": ['Group"]
      }
    }
  ]
}
```

注意

不支持在创建过程中运行并行条件。因此，请考虑根据可用的标准定义嵌套条件策略。

嵌套条件示例

```
{
  "anyOf": [
    {
      "rule": "IS_ENTITY_OWNER",
      "resourceType": "catalog-entity",
      "params": {
        "claims": ["group:default/team-a"]
      }
    },
    {
      "rule": "IS_ENTITY_KIND",
      "resourceType": "catalog-entity",
      "params": {
        "kinds": ['Group"]
      }
    }
  ],
  "not": {
    "rule": "IS_ENTITY_KIND",
    "resourceType": "catalog-entity",
    "params": { "kinds": ["Api"] }
  }
}
```

您可以通过添加额外的参数将前面的示例条件应用到 RBAC REST API，如下所示：

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/test",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["read"],
  "conditions": {
    "anyOf": [
      {
        "rule": "IS_ENTITY_OWNER",
        "resourceType": "catalog-entity",
        "params": {
          "claims": ["group:default/team-a"]
        }
      },
      {
        "rule": "IS_ENTITY_KIND",
        "resourceType": "catalog-entity",
        "params": {
          "kinds": ["Group"]
        }
      }
    ]
  }
}
```

以下示例可用于 Developer Hub 插件。这些示例可帮助您确定如何定义条件策略：

为 Keycloak 插件定义的条件策略

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/developer",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["update", "delete"],
  "conditions": {
    "not": {
      "rule": "HAS_ANNOTATION",
      "resourceType": "catalog-entity",
      "params": { "annotation": "keycloak.org/realm", "value": "<YOUR_REALM>" }
```

```
    }
  }
}
```

前面的 Keycloak 插件示例可防止 role:default/developer 中的用户更新或删除在 Keycloak 插件中放入目录的用户。

注意

在上例中，注解 keycloak.org/realm 需要 < YOUR_REALM > 的值。

为 Quay 插件定义的条件策略

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/developer",
  "pluginId": "scaffolder",
  "resourceType": "scaffolder-action",
  "permissionMapping": ["use"],
  "conditions": {
    "not": {
      "rule": "HAS_ACTION_ID",
      "resourceType": "scaffolder-action",
      "params": { "actionId": "quay:create-repository" }
    }
  }
}
```

前面的 Quay 插件示例可防止角色 role:default/developer 使用 Quay builder 操作。请注意，permissionMapping 包含使用 ，表示 scaffolder-action 资源类型权限没有权限策略。

有关 Red Hat Developer Hub 中权限的更多信息，请参阅 第 10.1 节 “Red Hat Developer Hub 中的权限策略”。

10.3. 使用 RED HAT DEVELOPER HUB WEB UI 管理基于角色的访问控制(RBAC)

管理员可以使用 Developer Hub Web 界面(Web UI)将特定的角色和权限分配给单独的用户或组。分配角色可确保在 Developer Hub 中规范对资源和功能的访问。

使用 Developer Hub 中的管理员角色，您可以为用户和组分配权限，允许用户或组使用 Developer Hub Web UI 查看、创建、修改和删除角色。

要访问 Web UI 中的 RBAC 功能，您必须安装并配置 @janus-idp/backstage-plugin-rbac 插件作为动态插件。有关安装动态插件的更多信息，请参阅在 Red Hat Developer Hub 中配置插件。

安装 @janus-idp/backstage-plugin-rbac 插件后，Administration 选项会出现在侧边栏的底部。当您可以点 Administration 时，默认为 RBAC 选项卡，显示 Developer Hub 中创建的所有现有角色。在 RBAC 选项卡中，您还可以查看与角色关联的权限策略总数。您还可以使用 Actions 列编辑或删除角色。

10.3.1. 在 Red Hat Developer Hub Web UI 中创建角色

您可以使用 Web UI 在 Red Hat Developer Hub 中创建角色。

先决条件

- 在 Developer Hub 中具有管理员角色。
- 您已在 Developer Hub 中安装了 @janus-idp/backstage-plugin-rbac 插件。如需更多信息，请参阅在 Red Hat Developer Hub 中配置插件。
- 您已配置了所需的权限策略。如需更多信息，请参阅 第 10.1.1 节 “权限策略配置”。

流程

1. 前往 Developer Hub 中边栏底部的 管理。

   此时会出现 RBAC 选项卡，在 Developer Hub 中显示所有创建的角色。

2. （可选）点击任何角色来查看 OVERVIEW 页面中的角色信息。

3. 单击 **CREATE** 以创建角色。

4. 在给定字段中输入角色的名称和描述，然后单击 **NEXT**。

5. 使用搜索字段添加用户和组，然后单击 **NEXT**。

6. 从 **Add permission policies** 部分的下拉菜单中选择 **Plugin** 和 **Permission**。

7. 选择或清除要在 **Add permission policies** 部分中设置的 **Policy**，然后单击 **NEXT**。

8. 查看 **Review and create** 部分中的添加的信息。

9. 点 **CREATE**。

验证

创建的角色会出现在 **RBAC** 选项卡中可用的列表中。

10.3.2. 在 Red Hat Developer Hub Web UI 中编辑角色

您可以使用 Web UI 编辑 Red Hat Developer Hub 中的角色。

注意

从 policy.csv 或 ConfigMap 文件生成的策略无法使用 Developer Hub Web UI 编辑或删除。

先决条件

- 在 Developer Hub 中具有管理员角色。
- 您已在 Developer Hub 中安装了 @janus-idp/backstage-plugin-rbac 插件。如需更多信息，请参阅在 Red Hat Developer Hub 中配置插件。

- 您已配置了所需的权限策略。如需更多信息，请参阅 第 10.1.1 节 “权限策略配置”。
- 您要编辑的角色在 Developer Hub 中创建。

流程

1. 前往 Developer Hub 中边栏底部的 管理。

   此时会出现 RBAC 选项卡，在 Developer Hub 中显示所有创建的角色。

2. （可选）点击任何角色来查看 OVERVIEW 页面中的角色信息。

3. 选择您要编辑的角色的编辑图标。

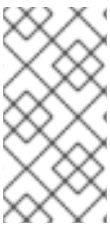
4. 编辑角色的详细信息，如名称、描述、用户和组以及权限策略，然后单击 NEXT。

5. 检查角色编辑的详细信息，然后单击 SAVE。

编辑角色后，您可以查看角色的 OVERVIEW 页面中 角色的 编辑详情。您还可以使用 OVERVIEW 页面上相应卡上的编辑图标来编辑角色的用户和组或权限。

10.3.3. 删除 Red Hat Developer Hub Web UI 中的角色

您可以使用 Web UI 删除 Red Hat Developer Hub 中的角色。



注意

从 policy.csv 或 ConfigMap 文件生成的策略无法使用 Developer Hub Web UI 编辑或删除。

先决条件

- 在 Developer Hub 中具有管理员角色。
- 您已在 Developer Hub 中安装了 @janus-idp/backstage-plugin-rbac 插件。如需更多信息，请参阅在 Red Hat Developer Hub 中配置插件。
- 您已配置了所需的权限策略。如需更多信息，请参阅 第 10.1.1 节 “权限策略配置”。
- 要删除的角色在 Developer Hub 中创建。

流程

1. 前往 Developer Hub 中边栏底部的 管理。

   此时会出现 RBAC 选项卡，在 Developer Hub 中显示所有创建的角色。

2. （可选）点击任何角色来查看 OVERVIEW 页面中的角色信息。

3. 在您要删除的角色的 Actions 列中选择删除图标。

   删除此角色？ 弹出窗口会出现在屏幕上。

4. 点 DELETE。

10.4. 基于角色的访问控制(RBAC) REST API

Red Hat Developer Hub 提供 RBAC REST API，可用于管理 Developer Hub 中的权限和角色。此 API 允许您促进和自动化 Developer Hub 权限策略和角色的维护。

使用 RBAC REST API，您可以执行以下操作：

- 检索有关所有权限策略或特定权限策略或角色的信息

- 创建、更新或删除权限策略或角色
- 检索有关静态插件的权限策略信息

RBAC REST API 需要以下组件：

授权

RBAC REST API 需要允许的用户角色的 Bearer 令牌授权。出于开发目的，您可以在浏览器中访问 Web 控制台。当您在网络请求列表中刷新令牌请求时，您可以在响应 JSON 中查找令牌。

```
Authorization: Bearer $token
```

例如，在 Developer Hub Homepage 中，导航到 Network 选项卡并搜索 query?term= 网络调用。或者，您也可以前往 Catalog 页面，再选择任何 Catalog API 网络调用来获取 Bearer 令牌。

HTTP 方法

RBAC REST API 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 ：更新资源
- DELETE ：删除资源

基本 URL

RBAC REST API 请求的基本 URL 是 http://SERVER:PORT/api/permission/policies，如 http://localhost:7007/api/permission/policies。

Endpoints

RBAC REST API 端点（如 /api/permission/policies/[kind]/[namespace]/[name]）用于指定 kind、namespace 和 name，是附加到基本 URL 的 URI，以访问对应的资源。

/api/permission/policies/[kind]/[namespace]/[name] 端点的请求 URL 示例：

http://localhost:7007/api/permission/policies/user/default/johndoe

注意

如果至少有一个权限分配给 user:default/johndoe，则前面提到的示例请求 URL 会返回带有有效授权令牌的 GET 响应中的结果。但是，如果只为角色分配权限，则示例请求 URL 不会返回输出。

请求数据

RBAC REST API 中的 HTTP POST 请求可能需要带有数据的 JSON 请求正文以及请求。

http://localhost:7007/api/permission/policies 的 POST 请求 URL 和 JSON 请求正文数据示例：

```
{
  "entityReference": "role:default/test",
  "permission": "catalog-entity",
  "policy": "delete",
  "effect": "allow"
}
```

HTTP 状态代码

RBAC REST API 支持以下 HTTP 状态代码来返回响应：

- 200 OK ：请求成功。
- 201 created ：请求成功创建新资源。
- 204 no Content: 请求成功，但响应有效负载中没有发送额外内容。

- 400 bad Request: input error with the request
- 401 未授权：缺少所请求资源的有效身份验证
- 403 forbidden：拒绝授权请求
- 404 not Found：无法找到请求的资源
- 409 冲突：请求与当前状态和目标资源冲突

Source

使用 RBAC 插件创建的每个权限策略和角色都与源关联，以维护插件中的数据一致性。您可以根据以下指定源信息操作权限策略和角色：

- CSV 文件
- 配置文件
- REST API
- Legacy

管理源自 CSV 文件和 REST API 的角色和权限策略涉及根据其初始源信息直接修改。

配置文件与 RBAC 插件提供的默认 role:default/rbac_admin 角色相关。默认角色具有创建、读取、更新和删除权限策略或角色以及读取目录实体的权限。

注意

如果您的管理要求默认权限不足，您可以创建具有所需权限策略的自定义 admin 角色。

传统源适用于在 RBAC 后端插件版本 2.1.3 之前定义的策略和角色，并且是源位置选项中最严格的限制。您必须更新旧源中的权限和角色，以使用 REST API 或 CSV 文件源。

您可以使用 GET 请求来查询角色和策略，并根据需要确定源信息。

10.4.1. 使用 REST 客户端或 curl 实用程序通过 RBAC REST API 发送请求

RBAC REST API 可让您在不使用用户界面的情况下与 Developer Hub 中的权限策略和角色交互。您可以使用任何 REST 客户端或 curl 实用程序发送 RBAC REST API 请求。

先决条件

- Red Hat Developer Hub 已安装并运行。有关安装 Red Hat Developer Hub 的更多信息，请参阅 第 2.1 节 “使用 Helm Chart 在 OpenShift Container Platform 上部署 Red Hat Developer Hub”。
- 您可以访问 Developer Hub。

流程

1. 识别您要发送请求的相关 API 端点，如 POST /api/permission/policies。根据您的用例调整任何请求详情。

   对于 REST 客户端 ：

   - Authorization ：从 web 控制台输入生成的令牌。
   - HTTP 方法：设置为 POST。
   -

URL ：输入 RBAC REST API 基础 URL 和端点，如 http://localhost:7007/api/permission/policies。

对于 curl 工具 ：

- -x: 设置为 POST
- - h ：设置以下标头：

  content-type: application/json

  Authorization: Bearer $token

  $token 是浏览器中从 Web 控制台请求的令牌。
- URL ：输入以下 RBAC REST API 基础 URL 端点，如 http://localhost:7007/api/permission/policies
- -d ：添加请求 JSON 正文

请求示例 ：

```
curl -X POST "http://localhost:7007/api/permission/policies" -d
'{"entityReference":"role:default/test", "permission": "catalog-entity", "policy": "read",
"effect":"allow"}' -H "Content-Type: application/json" -H "Authorization: Bearer $token" -
v
```

2. 执行请求并检查响应。

10.4.2. 支持的 RBAC REST API 端点

RBAC REST API 提供了在 Developer Hub 中管理角色、权限和条件策略的端点，并检索有关角色和策略的信息。

10.4.2.1. 角色

RBAC REST API 支持以下端点来管理 Red Hat Developer Hub 中的角色。

[GET] /api/permission/roles

返回 Developer Hub 中的所有角色。

响应示例(JSON)

```
[
  {
    "memberReferences": ["user:default/username"],
    "name": "role:default/guests"
  },
  {
    "memberReferences": [
      "group:default/groupname",
      "user:default/username"
    ],
    "name": "role:default/rbac_admin"
  }
]
```

[GET] /api/permission/roles/{kind}/{namespace}/{name}

返回 Developer Hub 中单个角色的信息。

响应示例(JSON)

```
[
  {
    "memberReferences": [
      "group:default/groupname",
      "user:default/username"
    ],
    "name": "role:default/rbac_admin"
  }
]
```

[POST] /api/permission/roles/{kind}/{namespace}/{name}

在 Developer Hub 中创建角色。

表 10.2. 请求参数

名称	描述	类型	存在
正文（body）	memberReferences、组、namespace，并 命名 要创建的新角色。	请求正文	必填

请求正文(JSON)示例

```
{
  "memberReferences": ["group:default/test"],
  "name": "role:default/test_admin"
}
```

响应示例

```
201 Created
```

[PUT] /api/permission/roles/{kind}/{namespace}/{name}

为 Developer Hub中的角色 更新 memberReferences、类型、命名空间或 名称。

请求参数

请求正文包含 oldRole 和 newRole 对象：

名称	描述	类型	存在
正文（body）	**memberReferences**、组、**namespace**，并 **命名** 要创建的新角色。	请求正文	必填

请求正文(JSON)示例

```
{
  "oldRole": {
    "memberReferences": ["group:default/test"],
    "name": "role:default/test_admin"
  },
  "newRole": {
    "memberReferences": ["group:default/test", "user:default/test2"],
    "name": "role:default/test_admin"
  }
}
```

响应示例

```
200 OK
```

[DELETE] /api/permission/roles/{kind}/{namespace}/{name}?memberReferences=<VALUE>

从 Developer Hub 中的角色中删除指定的用户或组。

表 10.3. 请求参数

名称	描述	类型	存在
kind	实体的类型	字符串	必填
namespace	实体的命名空间	字符串	必填
name	实体的名称	字符串	必填

名称	描述	类型	存在
memberReferences	关联的组信息	字符串	必填

响应示例

```
204
```

[DELETE] /api/permission/roles/{kind}/{namespace}/{name}

从 Developer Hub 中删除指定的角色。

表 10.4. 请求参数

名称	描述	类型	存在
kind	实体的类型	字符串	必填
namespace	实体的命名空间	字符串	必填
name	实体的名称	字符串	必填

响应示例

```
204
```

10.4.2.2. 权限策略

RBAC REST API 支持以下端点来管理 Red Hat Developer Hub 中的权限策略。

[GET] /api/permission/policies

返回所有用户的权限策略列表。

响应示例(JSON)

```
[
  {
    "entityReference": "role:default/test",
    "permission": "catalog-entity",
    "policy": "read",
    "effect": "allow",
    "metadata": {
      "source": "csv-file"
    }
  },
  {
    "entityReference": "role:default/test",
    "permission": "catalog.entity.create",
    "policy": "use",
    "effect": "allow",
    "metadata": {
      "source": "csv-file"
    }
  },
]
```

[GET] /api/permission/policies/{kind}/{namespace}/{name}

返回与指定实体引用相关的权限策略。

表 10.5. 请求参数

名称	描述	类型	存在
kind	实体的类型	字符串	必填
namespace	实体的命名空间	字符串	必填
name	与实体相关的名称	字符串	必填

响应示例(JSON)

```
[
  {
    "entityReference": "role:default/test",
    "permission": "catalog-entity",
    "policy": "read",
    "effect": "allow",
    "metadata": {
      "source": "csv-file"
    }
  },
  {
    "entityReference": "role:default/test",
    "permission": "catalog.entity.create",
    "policy": "use",
    "effect": "allow",
    "metadata": {
      "source": "csv-file"
    }
  }
]
```

[POST] /api/permission/policies

为**指定**实**体**创**建**权**限策略。**

表 10.6. 请求参数

名称	描述	类型	存在
entityReference	实体的引用值，包括 **类型**、**namespace** 和 **name**	字符串	必填
权限	特定插件、资源类型或名称的权限	字符串	必填
policy	权限的策略操作，如创建、**读取**、**更新**、**删除 或使用**	字符串	必填
effect	指明允许策略	字符串	必填

请求正文(JSON)示例

```
[
  {
    "entityReference": "role:default/test",
    "permission": "catalog-entity",
    "policy": "read",
    "effect": "allow"
  }
]
```

响应示例

```
201 Created
```

[PUT] /api/permission/policies/{kind}/{namespace}/{name}

更新指定实体的权限策略。

请求参数

请求正文包含 oldPolicy 和 newPolicy 对象：

名称	描述	类型	存在
权限	特定插件、资源类型或名称的权限	字符串	必填
policy	权限的策略操作，如创建、**读取**、**更新**、**删除**或**使用**	字符串	必填
effect	指明允许策略	字符串	必填

请求正文(JSON)示例

```
{
 "oldPolicy": [
  {
   "permission": "catalog-entity",
   "policy": "read",
   "effect": "allow"
  },
  {
   "permission": "catalog.entity.create",
   "policy": "create",
   "effect": "allow"
  }
 ],
 "newPolicy": [
  {
   "permission": "catalog-entity",
   "policy": "read",
   "effect": "deny"
  },
  {
   "permission": "policy-entity",
   "policy": "read",
   "effect": "allow"
  }
 ]
}
```

响应示例

```
200
```

[DELETE] /api/permission/policies/{kind}/{namespace}/{name}?permission={value1}&policy={value2}&effect={value3}

删除添加到指定实体的权限策略。

表 10.7. 请求参数

名称	描述	类型	存在
kind	实体的类型	字符串	必填
namespace	实体的命名空间	字符串	必填
name	与实体相关的名称	字符串	必填
权限	特定插件、资源类型或名称的权限	字符串	必填
policy	权限的策略操作，如创建、**读取**、**更新**、**删除 或使用**	字符串	必填
effect	指明允许策略	字符串	必填

响应示例

```
204 No Content
```

[DELETE] /api/permission/policies/{kind}/{namespace}/{name}

删除添加到指定实体的所有权限策略。

表 10.8. 请求参数

名称	描述	类型	存在
kind	实体的类型	字符串	必填
namespace	实体的命名空间	字符串	必填
name	与实体相关的名称	字符串	必填

响应示例

> 204 No Content

[GET] /api/permission/plugins/policies

返回所有静态插件的权限策略。

响应示例(JSON)


```
[
  {
    "pluginId": "catalog",
    "policies": [
      {
        "isResourced": true,
        "permission": "catalog-entity",
        "policy": "read"
      },
      {
        "isResourced": false,
        "permission": "catalog.entity.create",
        "policy": "create"
      },
      {
        "isResourced": true,
        "permission": "catalog-entity",
        "policy": "delete"
      },
      {
        "isResourced": true,
        "permission": "catalog-entity",
        "policy": "update"
      },
      {
        "isResourced": false,
        "permission": "catalog.location.read",
        "policy": "read"
      },
      {
        "isResourced": false,
        "permission": "catalog.location.create",
        "policy": "create"
      },
      {
        "isResourced": false,
```

```
      "permission": "catalog.location.delete",
      "policy": "delete"
    }
  ]
 },
 ...
]
```

10.4.2.3. 条件策略

RBAC REST API 支持以下端点来管理 Red Hat Developer Hub 中的条件策略。

[GET] /api/plugins/condition-rules

为 Developer Hub 中启用的可用插件返回可用条件规则参数模式。

响应示例(JSON)

```
[
  {
    "pluginId": "catalog",
    "rules": [
      {
        "name": "HAS_ANNOTATION",
        "description": "Allow entities with the specified annotation",
        "resourceType": "catalog-entity",
        "paramsSchema": {
          "type": "object",
          "properties": {
            "annotation": {
              "type": "string",
              "description": "Name of the annotation to match on"
            },
            "value": {
              "type": "string",
              "description": "Value of the annotation to match on"
            }
          },
          "required": [
            "annotation"
          ],
          "additionalProperties": false,
          "$schema": "http://json-schema.org/draft-07/schema#"
        }
      },
```

```
{
  "name": "HAS_LABEL",
  "description": "Allow entities with the specified label",
  "resourceType": "catalog-entity",
  "paramsSchema": {
    "type": "object",
    "properties": {
      "label": {
        "type": "string",
        "description": "Name of the label to match on"
      }
    },
    "required": [
      "label"
    ],
    "additionalProperties": false,
    "$schema": "http://json-schema.org/draft-07/schema#"
  }
},
{
  "name": "HAS_METADATA",
  "description": "Allow entities with the specified metadata subfield",
  "resourceType": "catalog-entity",
  "paramsSchema": {
    "type": "object",
    "properties": {
      "key": {
        "type": "string",
        "description": "Property within the entities metadata to match on"
      },
      "value": {
        "type": "string",
        "description": "Value of the given property to match on"
      }
    },
    "required": [
      "key"
    ],
    "additionalProperties": false,
    "$schema": "http://json-schema.org/draft-07/schema#"
  }
},
{
  "name": "HAS_SPEC",
  "description": "Allow entities with the specified spec subfield",
  "resourceType": "catalog-entity",
  "paramsSchema": {
    "type": "object",
    "properties": {
      "key": {
        "type": "string",
        "description": "Property within the entities spec to match on"
      },
      "value": {
        "type": "string",
        "description": "Value of the given property to match on"
```

```
        }
      },
      "required": [
        "key"
      ],
      "additionalProperties": false,
      "$schema": "http://json-schema.org/draft-07/schema#"
    }
  },
  {
    "name": "IS_ENTITY_KIND",
    "description": "Allow entities matching a specified kind",
    "resourceType": "catalog-entity",
    "paramsSchema": {
      "type": "object",
      "properties": {
        "kinds": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "description": "List of kinds to match at least one of"
        }
      },
      "required": [
        "kinds"
      ],
      "additionalProperties": false,
      "$schema": "http://json-schema.org/draft-07/schema#"
    }
  },
  {
    "name": "IS_ENTITY_OWNER",
    "description": "Allow entities owned by a specified claim",
    "resourceType": "catalog-entity",
    "paramsSchema": {
      "type": "object",
      "properties": {
        "claims": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "description": "List of claims to match at least one on within ownedBy"
        }
      },
      "required": [
        "claims"
      ],
      "additionalProperties": false,
      "$schema": "http://json-schema.org/draft-07/schema#"
    }
  }
]
```

```
    }
    ... <another plugin condition parameter schemas>
]
```

[GET] /api/permission/roles/conditions/:id

返回指定 ID 的条件。

响应示例(JSON)

```
{
  "id": 1,
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/test",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["read"],
  "conditions": {
    "anyOf": [
      {
        "rule": "IS_ENTITY_OWNER",
        "resourceType": "catalog-entity",
        "params": {
          "claims": ["group:default/team-a"]
        }
      },
      {
        "rule": "IS_ENTITY_KIND",
        "resourceType": "catalog-entity",
        "params": {
          "kinds": ["Group"]
        }
      }
    ]
  }
}
```

[GET] /api/permission/roles/conditions

返回所有角色的所有条件列表。

响应示例(JSON)

```
[
  {
    "id": 1,
    "result": "CONDITIONAL",
    "roleEntityRef": "role:default/test",
    "pluginId": "catalog",
    "resourceType": "catalog-entity",
    "permissionMapping": ["read"],
    "conditions": {
      "anyOf": [
        {
          "rule": "IS_ENTITY_OWNER",
          "resourceType": "catalog-entity",
          "params": {
            "claims": ["group:default/team-a"]
          }
        },
        {
          "rule": "IS_ENTITY_KIND",
          "resourceType": "catalog-entity",
          "params": {
            "kinds": ["Group"]
          }
        }
      ]
    }
  }
]
```


[POST] /api/permission/roles/conditions

为指定角色创建条件策略。

表 10.9. 请求参数

名称	描述	类型	存在
result	始终值为 **CONDITIONAL**	字符串	必填
roleEntityRef	对 RBAC 角色的字符串实体引用，如 **role:default/dev**	字符串	必填
pluginId	对应的插件 ID，如 **目录**	字符串	必填

名称	描述	类型	存在
permissionMapping	数组权限操作，如 **['read', 'update', 'delete']**	字符串数组	必填
resourceType	插件提供的资源类型，如 **catalog-entity**	字符串	必填
conditions	带有参数或数组参数的条件 JSON，按条件加入	JSON	必填
name	角色的名称	字符串	必填
metadata.description	角色的描述	字符串	选填

请求正文(JSON)示例

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/test",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["read"],
  "conditions": {
    "rule": "IS_ENTITY_OWNER",
    "resourceType": "catalog-entity",
    "params": {
      "claims": ["group:default/team-a"]
    }
  }
}
```

响应示例(JSON)

```
{
  "id": 1
}
```

[PUT] /permission/roles/conditions/:id

更新指定 ID 的条件策略。

表 10.10. 请求参数

名称	描述	类型	存在
result	始终值为 **CONDITIONAL**	字符串	必填
roleEntityRef	对 RBAC 角色的字符串实体引用，如 **role:default/dev**	字符串	必填
pluginId	对应的插件 ID，如 **目录**	字符串	必填
permissionMapping	数组权限操作，如 **['read', 'update', 'delete']**	字符串数组	必填
resourceType	插件提供的资源类型，如 **catalog-entity**	字符串	必填
conditions	带有参数或数组参数的条件 JSON，按条件加入	JSON	必填
name	角色的名称	字符串	必填
metadata.description	角色的描述	字符串	选填

请求正文(JSON)示例

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/test",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["read"],
  "conditions": {
    "anyOf": [
      {
        "rule": "IS_ENTITY_OWNER",
        "resourceType": "catalog-entity",
        "params": {
          "claims": ["group:default/team-a"]
        }
      },
      {
```

```
        "rule": "IS_ENTITY_KIND",
        "resourceType": "catalog-entity",
        "params": {
          "kinds": ['Group']
        }
      }
    ]
  }
}
```

响应示例

```
200
```

[DELETE] /api/permission/roles/conditions/:id

删除指定 ID 的条件策略。

响应示例

```
204
```

第 11 章 管理模板

模板是由 YAML 文件中定义的不同 UI 字段组成的形式。模板包括 *操作*，它们是按顺序执行的步骤，并可有条件地执行。

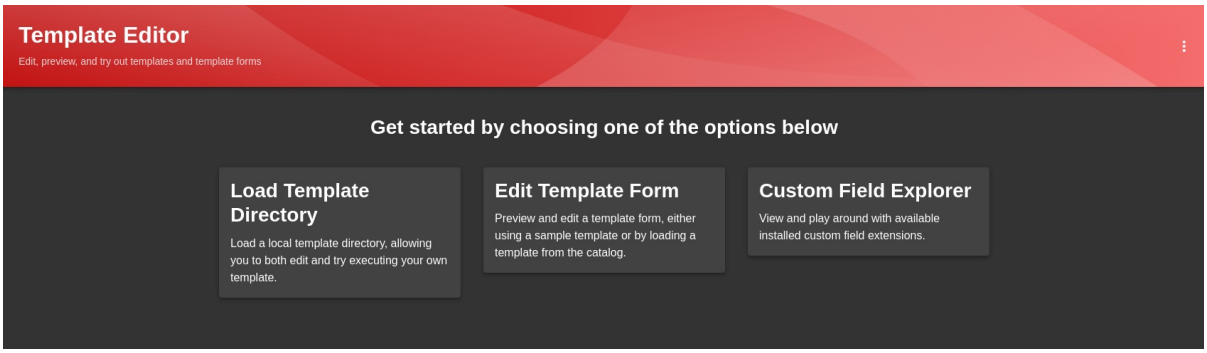
您可以使用模板轻松创建 Red Hat Developer Hub 组件，然后将这些组件发布到不同的位置，如 Red Hat Developer Hub 软件目录或 GitHub 或 GitLab 中的存储库。

11.1. 使用 TEMPLATE EDITOR 创建模板

您可以使用 Template Editor 创建模板。

流程

1. 使用以下选项之一访问 Template Editor：



   - 为您的 Red Hat Developer Hub 实例打开 URL https://<rhdh_url>/create/edit。
   - 在 Red Hat Developer Hub 控制台的导航菜单中点 Create...，然后点 overflow 菜单按钮并选择 Template 编辑器。

2. 点 Edit Template Form。
3. 可选：修改模板参数的 YAML 定义。有关这些参数的详情请参考 第 11.2 节 “创建模板作为 YAML 文件”。
4. 在 Name * 字段中，为模板输入一个唯一名称。

5. 从 Owner 下拉菜单中选择模板的所有者。

6. 点击 Next。

7. 在 Repository Location 视图中，输入您要发布模板的托管存储库的以下信息：

   a. 从下拉菜单中选择 可用主机。

   注意

   可用主机通过 allowedHosts 字段在 YAML 参数中定义：

   YAML 示例

   ```
   # ...
         ui:options:
           allowedHosts:
             - github.com
   # ...
   ```

   b. 在 Owner * 字段中，输入托管存储库所属的机构、用户或项目。

   c. 在 Repository * 字段中，输入托管存储库的名称。

   d. 点 Review。

8. 检查信息准确性，然后点 Create。

验证

1. 单击导航面板中的 Catalog 选项卡。

2. 在 Kind 下拉菜单中选择 Template。

3. 确认模板显示在现有模板列表中。

11.2. 创建模板作为 YAML 文件

您可以通过将 Template 对象定义为 YAML 文件来创建模板。

Template 对象描述了模板及其元数据。它还包含必要的输入变量以及由 scaffolding 服务执行的操作列表。

模板 对象示例

```
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

指定模板的名称。

2

指定模板的标题。这是在 Create… 视图中的模板标题上可见的标题。

3

指定模板的描述。这是在 Create… view 中的模板标题上可见的描述。

4

指定模板的所有权。owner 字段提供有关负责维护或监督系统或机构中模板的信息。在提供的示例中，owner 字段设置为 backstage/busybox-core。这意味着此模板属于 backstage 命名空间中的 zFCP -core 项目。

5

指定组件类型。此必填字段接受任何字符串值，但您的组织应为其建立正确的税款。Red Hat Developer Hub 实例可能会读取此字段，并根据它的值的不同行为。例如，Web 站点 类型组件可能会在特定于 Web 站点的 Red Hat Developer Hub 界面中显示工具。

此字段通常有以下值：

service

后端服务，通常是公开 API。

网站

网站。

程序库

软件库，如 npm 模块或 Java 库。

6

当用户使用 Red Hat Developer Hub 控制台中的模板创建组件时，使用 parameters 部分为用户输入指定参数。每个 参数 子部分由标题和属性定义，创建一个具有该定义的新表单页面。

7

使用 steps 部分指定后端中执行的步骤。这些步骤必须通过使用唯一的步骤 ID、名称和操作来定义。您可以通过访问 URL https://<rhdh_url>/create/actions 来查看 Red Hat Developer Hub 实例上可用的操作。

8

使用 output 部分指定使用模板时创建的输出数据的结构。output 部分（特别是 links 子部分）为用户提供了可用于访问和与从模板创建的组件交互的宝贵引用和 URL。

9

提供与生成组件关联的存储库的引用或 URL。

10

提供允许用户在列出各种组件的目录或目录中打开生成的组件的参考或 URL。

其他资源

- Backstage 文档 - 编写模板
- Backstage 文档 - 内置操作
- Backstage 文档 - 编写自定义操作

11.3. 将现有模板导入到 RED HAT DEVELOPER HUB

您可以使用 Catalog Processor 在 Red Hat Developer Hub 实例中添加现有模板。

先决条件

- 您已创建了一个目录或仓库，其中包含至少一个模板 YAML 文件。
- 如果要使用存储在 GitHub 或 GitLab 等存储库中的模板，您必须为您的供应商配置 Red Hat Developer Hub 集成。

流程

- 在 app-config.yaml 配置文件中，修改 catalog.rules 部分使其包含模板的规则，并将 catalog.locations 部分配置为指向要添加的模板，如下例所示：

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

1. 要允许新模板添加到目录中，您必须添加一个 Template 规则。
2. 如果您要从存储库（如 GitHub 或 GitLab）导入模板，请使用 url 类型。
3. 指定模板的 URL。

验证

1. 单击导航面板中的 Catalog 选项卡。

2. 在 Kind 下拉菜单中选择 Template。

3. 确认模板显示在现有模板列表中。

其他资源

- 在 Developer Hub 中配置 GitHub 应用程序
- 启用 GitLab OAuth 身份验证供应商

第 12 章 在 RED HAT DEVELOPER HUB 中配置 TECHDOCS 插件

Red Hat Developer Hub TechDocs 插件可帮助您的组织在中央位置创建、查找和使用文档，并以标准化的方式使用。例如：

文档类代码方法

在 Markdown 文件中编写存储在项目存储库中的技术文档以及您的代码。

文档站点生成

使用 MkDocs 为您的在 Developer Hub 中呈现的文档创建一个功能全面、基于 Markdown 的静态 HTML 站点。

文档站点元数据和集成

请参阅有关文档站点的额外元数据以及静态文档，如最后一次更新的日期、站点所有者、顶级贡献者、打开 GitHub 问题、Slack 支持频道和 Stack Overflow Enterprise 标签。

内置导航和搜索

从文档中更快、更轻松地查找您想要的信息。

附加组件

使用附加组件自定义 TechDocs 体验，以解决高顺序的文档需求。

TechDocs 插件是预安装并在 Developer Hub 实例上启用的。您可以通过配置 Red Hat Developer Hub Helm chart 或 Red Hat Developer Hub Operator 配置映射来禁用或启用 TechDocs 插件并更改其他参数。

重要

Red Hat Developer Hub 包括一个内置的 TechDocs 构建器，它从您的代码库中生成静态 HTML 文档。但是，本地构建器的默认基本设置不适用于生产环境。

您可以将 CI/CD 管道与具有专用作业的存储库搭配使用，为 TechDocs 生成文档。生成的静态文件存储在 OpenShift Data Foundation 中，或存储在您选择的云存储解决方案中，并发布到静态 HTML 文档站点。

将 OpenShift Data Foundation 配置为存储 TechDocs 生成的文件后，您可以配置 TechDocs 插件，以使用 OpenShift Data Foundation 进行云存储。

其他资源

- 如需更多信息，请参阅在 Red Hat Developer Hub 中配置插件。

12.1. 为 TECHDOCS 文件配置存储

TechDocs 发布程序将生成的文件存储在本地存储或云存储中，如 OpenShift Data Foundation、Google GCS、AWS S3 或 Azure Blob Storage。

12.1.1. 使用 OpenShift Data Foundation 进行文件存储

您可以配置 OpenShift Data Foundation 以存储 TechDocs 生成的文件，而不依赖于其他云存储解决方案。

OpenShift Data Foundation 提供了一个 ObjectBucketClaim 自定义资源(CR)，您可以使用它来请求 S3 兼容存储桶后端。您必须安装 OpenShift Data Foundation Operator 来使用此功能。

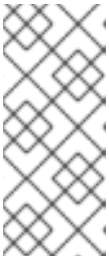
先决条件

- OpenShift Container Platform 管理员已在 Red Hat OpenShift Container Platform 中安装了 OpenShift Data Foundation Operator。如需更多信息，请参阅 OpenShift Container Platform - 安装 Red Hat OpenShift Data Foundation Operator。
- OpenShift Container Platform 管理员已创建了 OpenShift Data Foundation 集群并配置了 StorageSystem 模式。如需更多信息，请参阅 OpenShift Container Platform - 创建 OpenShift Data Foundation 集群。

流程

- 创建一个 ObjectBucketClaim CR，其中存储了生成的 TechDocs 文件。例如：

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: <rhdh_bucket_claim_name>
spec:
  generateBucketName: <rhdh_bucket_claim_name>
  storageClassName: openshift-storage.noobaa.io
```



注意

创建 Developer Hub ObjectBucketClaim CR 会自动创建 Developer Hub ObjectBucketClaim 配置映射和 secret。配置映射和 secret 的名称与 ObjetBucketClaim CR 的名称相同。

创建 ObjectBucketClaim CR 后，您可以使用配置映射和 secret 中存储的信息，使 Developer Hub 容器可以作为环境变量访问这些信息。根据用来安装 Developer Hub 的方法，您可以将访问信息添加到 Red Hat Developer Hub Helm Chart 或 Operator 配置中。

其他资源

- 如需有关对象 Bucket 声明的更多信息，请参阅 OpenShift Container Platform - Object Bucket Claim。

12.1.2. 使用 Helm Chart 使对象存储可供容器访问

创建 ObjectBucketClaim 自定义资源(CR)会自动生成 Developer Hub ObjectBucketClaim 配置映射和 secret。配置映射和 secret 包含 ObjectBucket 访问信息。在 Helm Chart 配置中添加访问信息可使 Developer Hub 容器访问它，方法是将以下环境变量添加到容器中：

- BUCKET_NAME
- BUCKET_HOST
- BUCKET_PORT
- BUCKET_REGION
- BUCKET_SUBREGION
- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY

然后，在 TechDocs 插件配置中使用这些变量。

先决条件

- 已使用 Helm Chart 在 OpenShift Container Platform 上安装 Red Hat Developer Hub。
- 您已创建了 ObjectBucketClaim CR 来存储 TechDocs 生成的文件。如需更多信息，请参阅使用 OpenShift Data Foundation 进行文件存储

流程

- 在 Helm Chart 值中的 upstream.backstage 键中，输入 Developer Hub ObjectBucketClaim secret 的名称，作为 extraEnvVarsSecrets 字段的值和 extraEnvVarsCM 字段。例如：

```
upstream:
  backstage:
    extraEnvVarsSecrets:
      - <rhdh_bucket_claim_name>
    extraEnvVarsCM:
      - <rhdh_bucket_claim_name>
```

12.1.2.1. Helm Chart 的 TechDocs 插件配置示例

以下示例显示了 TechDocs 插件的 Developer Hub Helm Chart 配置：

```
global:
  dynamic:
    includes:
      - 'dynamic-plugins.default.yaml'
  plugins:
    - disabled: false
      package: ./dynamic-plugins/dist/backstage-plugin-techdocs-backend-dynamic
      pluginConfig:
        techdocs:
          builder: external
          generator:
            runIn: local
          publisher:
            awsS3:
              bucketName: '${BUCKET_NAME}'
              credentials:
                accessKeyId: '${AWS_ACCESS_KEY_ID}'
                secretAccessKey: '${AWS_SECRET_ACCESS_KEY}'
              endpoint: 'https://${BUCKET_HOST}'
```

```
      regions: '${BUCKET_REGION}'
      s3ForcePathStyle: true
    type: awsS3
```

12.1.3. 使用 Operator 使容器可以访问对象存储

创建 ObjectBucketClaim 自定义资源(CR)会自动生成 Developer Hub ObjectBucketClaim 配置映射和 secret。配置映射和 secret 包含 ObjectBucket 访问信息。在 Operator 配置中添加访问信息可使 Developer Hub 容器访问它，方法是将以下环境变量添加到容器中：

- BUCKET_NAME
- BUCKET_HOST
- BUCKET_PORT
- BUCKET_REGION
- BUCKET_SUBREGION
- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY

然后，在 TechDocs 插件配置中使用这些变量。

先决条件

- 已使用 Operator 在 OpenShift Container Platform 上安装 Red Hat Developer Hub。
- 您已创建了 ObjectBucketClaim CR 来存储 TechDocs 生成的文件。

流程

- 在 Developer Hub ObjectBucketClaim CR 中，输入 Developer Hub ObjectBucketClaim 配置映射的名称作为 spec.application.extraEnvs.configMaps 字段的值，并输入 Developer Hub ObjectBucketClaim secret 名称作为 spec.application.extraEnvs.secrets 字段的值。例如：

```
spec:
  application:
    extraEnvs:
      configMaps:
        - name: <rhdh_bucket_claim_name>
      secrets:
        - name: <rhdh_bucket_claim_name>
```

12.1.3.1. Operator 的 TechDocs 插件配置示例

以下示例显示了 TechDocs 插件的 Red Hat Developer Hub Operator 配置映射配置：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: dynamic-plugins-rhdh
data:
  dynamic-plugins.yaml: |
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - disabled: false
        package: ./dynamic-plugins/dist/backstage-plugin-techdocs-backend-dynamic
        pluginConfig:
          techdocs:
            builder: external
            generator:
              runIn: local
            publisher:
              awsS3:
                bucketName: '${BUCKET_NAME}'
                credentials:
                  accessKeyId: '${AWS_ACCESS_KEY_ID}'
                  secretAccessKey: '${AWS_SECRET_ACCESS_KEY}'
                endpoint: 'https://${BUCKET_HOST}'
                regions: '${BUCKET_REGION}'
                s3ForcePathStyle: true
              type: awsS3
```

12.2. 配置 CI/CD 以生成和发布 TECDOCS 站点

xmlrpc 从云存储桶（如 OpenShift Data Foundation）读取静态生成的文档文件。文档站点在与包含文档文件的存储库关联的 CI/CD 工作流上生成。您可以使用 HEKETI -cli CLI 工具在 CI 上生成文档 并发

布到云存储。

您可以使用以下示例为 TechDocs 出版物创建一个脚本：

```
# Prepare
REPOSITORY_URL='https://github.com/org/repo'
git clone $REPOSITORY_URL
cd repo

# Install @techdocs/cli, mkdocs and mkdocs plugins
npm install -g @techdocs/cli
pip install "mkdocs-techdocs-core==1.*"

# Generate
techdocs-cli generate --no-docker

# Publish
techdocs-cli publish --publisher-type awsS3 --storage-name <bucket/container> --entity
<Namespace/Kind/Name>
```

当用户在包含文档文件的存储库中进行更改时，TechDocs 工作流会启动 CI。您只能将工作流配置为仅在 docs/ 目录或 mkdocs.yml 中的文件改变时启动。

12.2.1. 为 CI 准备存储库

CI 的第一步是将文档源存储库克隆到工作目录中。

流程

- 要在工作目录中克隆您的文档源存储库，请输入以下命令：

```
git clone <https://path/to/docs-repository/>
```

12.2.2. 生成 TechDocs 网站

流程

要将 CI/CD 配置为生成 HEKETI，请完成以下步骤：

1. 使用以下命令安装 npx 软件包 以运行 HEKETI-cli ：

```
npm install -g npx
```

2. **使用以下命令安装 HEKETI-cli 工具：**

```
npm install -g @techdocs/cli
```

3. **使用以下命令安装 mkdocs 插件：**

```
pip install "mkdocs-techdocs-core==1.*"
```

4. **使用以下命令生成 HEKETI 站点：**

```
npx @techdocs/cli generate --no-docker --source-dir <path_to_repo> --output-dir ./site
```

其中 < path_to_repo > 是用来克隆存储库的文件路径中的位置。

12.2.3. 发布 TechDocs 网站

流程

要发布您的 HEKETI 网站，请完成以下步骤：

1. 为您的云存储供应商设置必要的身份验证环境变量。

2. **使用以下命令发布您的 HEKETI ：**

```
npx @techdocs/cli publish --publisher-type <awsS3|googleGcs> --storage-name
<bucket/container> --entity <namespace/kind/name> --directory ./site
```

3. **在软件模板中添加 .github/workflows/wagon.yml 文件。例如：**

```
name: Publish TechDocs Site

on:
 push:
   branches: [main]
   # You can even set it to run only when TechDocs related files are updated.
   # paths:
```

```
    #   - "docs/**"
    #   - "mkdocs.yml"

jobs:
 publish-techdocs-site:
   runs-on: ubuntu-latest

   # The following secrets are required in your CI environment for publishing files to AWS S3.
   # e.g. You can use GitHub Organization secrets to set them for all existing and new
repositories.
   env:
     TECHDOCS_S3_BUCKET_NAME: ${{ secrets.TECHDOCS_S3_BUCKET_NAME }}
     AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
     AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
     AWS_REGION: ${{ secrets.AWS_REGION }}
     ENTITY_NAMESPACE: 'default'
     ENTITY_KIND: 'Component'
     ENTITY_NAME: 'my-doc-entity'
     # In a Software template, Scaffolder will replace {{cookiecutter.component_id | jsonify}}
     # with the correct entity name. This is same as metadata.name in the entity's catalog-
info.yaml
     # ENTITY_NAME: '{{ cookiecutter.component_id | jsonify }}'

   steps:
     - name: Checkout code
       uses: actions/checkout@v3

     - uses: actions/setup-node@v3
     - uses: actions/setup-python@v4
       with:
         python-version: '3.9'

     - name: Install techdocs-cli
       run: sudo npm install -g @techdocs/cli

     - name: Install mkdocs and mkdocs plugins
       run: python -m pip install mkdocs-techdocs-core==1.*

     - name: Generate docs site
       run: techdocs-cli generate --no-docker --verbose

     - name: Publish docs site
       run: techdocs-cli publish --publisher-type awsS3 --storage-name
$TECHDOCS_S3_BUCKET_NAME --entity
$ENTITY_NAMESPACE/$ENTITY_KIND/$ENTITY_NAME
```