# Red Hat Developer Hub 1.5

## 配置动态插件

在 Red Hat Developer Hub 中配置动态插件

Last Updated: 2025-07-16

# Red Hat Developer Hub 1.5 配置动态插件

在 Red Hat Developer Hub 中配置动态插件

## 法律通告



## 摘要

作为平台工程师，您可以在 RHDH 中配置动态插件，以访问您的开发基础架构或软件开发工具。

# 目录

# 第 1 章 为 RED HAT DEVELOPER HUB 安装 ANSIBLE 插件

Red Hat Developer Hub 的 Ansible 插件通过策展的学习路径、一键式内容创建、集成开发工具和其他建议资源提供 Ansible 特定的门户体验。

要安装和配置 Ansible 插件，*请参阅为 Red Hat Developer Hub 安装 Ansible 插件* 。

# 第 2 章 启用 ARGO CD 插件

您可以使用 Argo CD 插件来视觉化 OpenShift GitOps 中的持续交付(CD)工作流。此插件提供对应用的状态、部署详情、提交消息、提交者、提升到环境和部署历史记录的容器镜像的可视化概述。

**先决条件**

- 在 **app-config.yaml** configmap 中添加 Argo CD 实例信息，如下例所示：

```
argocd:
  appLocatorMethods:
    - type: 'config'
      instances:
        - name: argoInstance1
          url: https://argoInstance1.com
          username: ${ARGOCD_USERNAME}
          password: ${ARGOCD_PASSWORD}
        - name: argoInstance2
          url: https://argoInstance2.com
          username: ${ARGOCD_USERNAME}
          password: ${ARGOCD_PASSWORD}
```

- 将以下注解添加到实体的 **catalog-info.yaml** 文件中，以识别 Argo CD 应用程序。

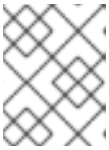
```
annotations:
  ...
  # The label that Argo CD uses to fetch all the applications. The format to be used is
label.key=label.value. For example, rht-gitops.com/janus-argocd=quarkus-app.

  argocd/app-selector: '${ARGOCD_LABEL_SELECTOR}'
```

- （可选）在实体的 **catalog-info.yaml** 文件中添加以下注解，以在 Argo CD 实例间切换，如下例所示：

```
annotations:
  ...
  # The Argo CD instance name used in `app-config.yaml`.

  argocd/instance-name: '${ARGOCD_INSTANCE}'
```



**注意**

如果没有设置此注解，Argo CD 插件默认为 **app-config.yaml** 中配置的第一个 Argo CD 实例。

**流程**

1. 将以下内容添加到 dynamic-plugins ConfigMap 中以启用 Argo CD 插件。

```
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
```

```
plugins:
  - package: ./dynamic-plugins/dist/roadiehq-backstage-plugin-argo-cd-backend-dynamic
    disabled: false
  - package: ./dynamic-plugins/dist/backstage-community-plugin-redhat-argocd
    disabled: false
```

## 2.1. 启用 ARGO CD ROLLOUTS

可选的 Argo CD Rollouts 功能通过为应用程序提供高级部署策略（如蓝绿和 canary 部署）来增强 Kubernetes。当集成到 backstage Kubernetes 插件时，它允许开发人员和运维团队在 Backstage 接口中无缝视觉化和管理 Argo CD Rollouts。

**先决条件**

- 已安装并配置 Backstage Kubernetes 插件(**@backstage/plugin-kubernetes**)。
  - 要在 Backstage 中安装和配置 Kubernetes 插件，请参阅 安装和配置指南。https://backstage.io/docs/features/kubernetes/installation/
- 您可以使用所需权限访问 Kubernetes 集群，以创建和管理自定义资源和 **ClusterRole**。
- Kubernetes 集群安装了 **argoproj.io** 组资源（如 Rollouts 和 AnalysisRuns）。

**流程**

1. 在 Backstage 实例的 **app-config.yaml** 文件中，在 **kubernetes** 配置下添加以下 **customResources** 组件以启用 Argo Rollouts 和 AnalysisRuns：

```
kubernetes:
  ...
  customResources:
    - group: 'argoproj.io'
      apiVersion: 'v1alpha1'
      plural: 'Rollouts'
    - group: 'argoproj.io'
      apiVersion: 'v1alpha1'
      plural: 'analysisruns'
```

2. 为自定义资源授予 **ClusterRole** 权限。

**注意**

- 如果已经配置了 Backstage Kubernetes 插件，则可能已经授予 Rollouts 和 AnalysisRuns 的 **ClusterRole** 权限。
- 使用 准备的清单 为 Kubernetes 和 ArgoCD 插件提供只读 **ClusterRole** 访问权限。

a. 如果没有授予 **ClusterRole** 权限，请使用以下 YAML 清单来创建 **ClusterRole** ：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: backstage-read-only
```

```
rules:
  - apiGroups:
      - argoproj.io
    resources:
      - rollouts
      - analysisruns
    verbs:
      - get
      - list
```

a. 使用 **kubectl** 将清单应用到集群：

```
kubectl apply -f <your-clusterrole-file>.yaml
```

b. 确保访问集群的 **ServiceAccount** 已分配此 **ClusterRole**。

3. 为 **catalog-info.yaml** 添加注解来标识 Backstage 的 Kubernetes 资源。

a. 通过实体 ID 识别资源：

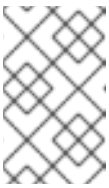
```
annotations:
  ...
  backstage.io/kubernetes-id: <BACKSTAGE_ENTITY_NAME>
```

b. （可选） 用于通过命名空间识别资源：

```
annotations:
  ...
  backstage.io/kubernetes-namespace: <RESOURCE_NAMESPACE>
```

c. 要使用自定义标签选择器，它覆盖由实体 ID 或命名空间的资源标识：

```
annotations:
  ...
  backstage.io/kubernetes-label-selector: 'app=my-app,component=front-end'
```



**注意**

确保指定 Kubernetes 资源上的 **backstage.io/kubernetes-label-selector** 中声明的标签。此注解会覆盖基于实体或基于命名空间的识别注解，如 **backstage.io/kubernetes-id** 和 **backstage.io/kubernetes-namespace**。

4. 为 Kubernetes 资源添加标签，以启用 Backstage 来查找适当的 Kubernetes 资源。

a. Backstage Kubernetes 插件标签：添加此标签以将资源映射到特定的 Backstage 实体。

```
labels:
  ...
  backstage.io/kubernetes-id: <BACKSTAGE_ENTITY_NAME>
```

b. GitOps 应用程序映射：添加此标签以将 Argo CD Rollouts 映射到特定的 GitOps 应用程序

```
labels:
  ...
  app.kubernetes.io/instance: <GITOPS_APPLICATION_NAME>
```

**注意**

如果使用标签选择器注解(backstage.io/kubernetes-label-selector)，请确保资源上存在指定的标签。标签选择器将覆盖其他注解，如 kubernetes-id 或 kubernetes-namespace。

**验证**

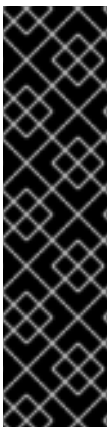
1. 将更新的配置推送到 GitOps 存储库，以触发推出部署。
2. 打开 Red Hat Developer Hub 界面并导航到您配置的实体。
3. 选择 **CD** 选项卡，然后选择 **GitOps 应用程序**。侧边面板将打开。
4. 在侧面板的 **Resources** 表中，验证是否显示以下资源：
   - rollouts
   - AnalysisRuns （可选）
5. 扩展推出部署资源并查看以下详情：
   - Revisions 行显示不同推出部署版本的流量分布详情。
   - Analysis Runs 行显示评估推出成功分析任务的状态。

**其他资源**

- 从 1.2 开始，Red Hat ArgoCD 插件的软件包路径、范围和名称已更改。如需更多信息，请参阅 *Red Hat Developer Hub* 发行注记中的 明细更改。
- 有关安装动态插件的更多信息，请参阅在 Red Hat Developer Hub 中安装和查看插件 。

# 第 3 章 安装和配置 JFROG ARTIFACTORY 插件

JFrog Artifactory 是一个前端插件，显示您存储在 JFrog Artifactory 存储库中的容器镜像的信息。JFrog Artifactory 插件预安装了 Developer Hub 并默认禁用。要使用它，您需要首先启用并配置它。



**重要**

JFrog Artifactory 插件只是一个技术预览功能。

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

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

Red Hat Developer Support Policy 页中提供了有关红帽如何提供对捆绑社区动态插件的支持的更多详情。

## 3.1. 安装

在 Developer Hub 中预装了 JFrog Artifactory 插件，具有基本配置属性。要启用它，请将 disabled 属性设置为 **false**，如下所示：

```
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: ./dynamic-plugins/dist/backstage-community-plugin-jfrog-artifactory
        disabled: false
```

## 3.2. 配置

1. 将代理设置为 **app-config.yaml** 文件中所需的 JFrog Artifactory 服务器，如下所示：

```
proxy:
  endpoints:
    '/jfrog-artifactory/api':
      target: http://<hostname>:8082 # or https://<customer>.jfrog.io
      headers:
      # Authorization: 'Bearer <YOUR TOKEN>'
      # Change to "false" in case of using a self-hosted Artifactory instance with a self-signed certificate
      secure: true
```

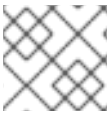
2. 在实体的 **catalog-info.yaml** 文件中添加以下注解，以启用 RHDH 组件中的 JFrog Artifactory 插件功能：

```
metadata:
   annotations:
     'jfrog-artifactory/image-name': '<IMAGE-NAME>'
```

# 第 4 章 安装和配置 KEYCLOAK

将 Keycloak 集成到 Developer Hub 的 Keycloak 后端插件具有以下功能：

- 在域中同步 Keycloak 用户。
- 在域中同步 Keycloak 组及其用户。



**注意**

支持的 Red Hat Build of Keycloak (RHBK)版本为 **24.0**。

## 4.1. 安装

Keycloak 插件在 Developer Hub 中预加载，具有基本配置属性。要启用它，请将 **disabled** 属性设置为 **false**，如下所示：

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

## 4.2. 基本配置

要启用 Keycloak 插件，您必须设置以下环境变量：

- **KEYCLOAK_BASE_URL**
- **KEYCLOAK_LOGIN_REALM**
- **KEYCLOAK_REALM**
- **KEYCLOAK_CLIENT_ID**
- **KEYCLOAK_CLIENT_SECRET**

## 4.3. 高级配置

### 调度配置

您可以在 **app-config.yaml** 文件中配置调度，如下所示：

```
    catalog:
      providers:
        keycloakOrg:
          default:
            # ...
            # highlight-add-start
            schedule: # optional; same options as in TaskScheduleDefinition
```

```
    # supports cron, ISO duration, "human duration" as used in code
    frequency: { minutes: 1 }
    # supports ISO duration, "human duration" as used in code
    timeout: { minutes: 1 }
    initialDelay: { seconds: 15 }
    # highlight-add-end
```

**注意**

如果您在 **app-config.yaml** 文件中对调度进行任何更改，请重新启动以应用更改。

### Keycloak 查询参数

您可以覆盖 **app-config.yaml** 文件中的默认 Keycloak 查询参数，如下所示：

```
catalog:
  providers:
    keycloakOrg:
      default:
        # ...
        # highlight-add-start
        userQuerySize: 500 # Optional
        groupQuerySize: 250 # Optional
        # highlight-add-end
```

使用 Keycloak API 启用 Developer Hub 和 Keycloak 之间的通信。用户名和密码或客户端凭证是支持的身份验证方法。

下表描述了您可以在 **app-config.yaml** 文件中的 **catalog.providers.keycloakOrg.<ENVIRONMENT_NAME>** 对象中启用插件的参数：

| Name | 描述 | 默认值 | 必填 |
|---|---|---|---|
| **baseUrl** | Keycloak 服务器的位置，如 **https://localhost:8443/auth**。 | "" | 是 |
| **realm** | 要同步的域 | **master** | 否 |
| **loginRealm** | 用于验证的域 | **master** | 否 |
| **username** | 用于身份验证的用户名 | "" | 如果使用基于密码的身份验证，则为 |
| **password** | 进行验证的密码 | "" | 如果使用基于密码的身份验证，则为 |
| **clientId** | 进行身份验证的客户端 ID | "" | 如果使用基于客户端凭证的身份验证，则为 |

| Name | 描述 | 默认值 | 必填 |
| --- | --- | --- | --- |
| **clientSecret** | 要进行身份验证的客户端 Secret | "" | 如果使用基于客户端凭证的身份验证，则为 |
| **userQuerySize** | 一次要查询的用户数 | **100** | 否 |
| **groupQuerySize** | 一次要查询的组数 | **100** | 否 |

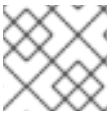
使用客户端凭据时，访问类型必须设置为 **confidential**，并且必须启用服务帐户。您还必须从 **realm-management** 客户端角色中添加以下角色：

- **query-groups**
- **query-users**
- **view-users**

## 4.4. 限制：

如果您有自签名或公司证书问题，您可以在启动 Developer Hub 前设置以下环境变量：

**NODE_TLS_REJECT_UNAUTHORIZED=0**

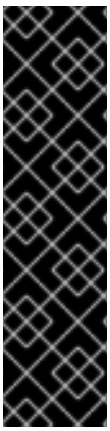


**注意**

不建议设置环境变量的解决方案。

# 第 5 章 安装和配置 NEXUS 存储库管理器插件

Nexus Repository Manager 插件显示 Developer Hub 应用中构建工件的信息。构建工件在 Nexus 存储库管理器中提供。



**重要**

Nexus 存储库管理器插件只是一个技术预览功能。

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

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

Red Hat Developer Support Policy 页中提供了有关红帽如何提供对捆绑社区动态插件的支持的更多详情。

## 5.1. 安装

Nexus 存储库管理器插件在 Developer Hub 中预加载，具有基本配置属性。要启用它，请将 disabled 属性设置为 **false**，如下所示：

```
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: ./dynamic-plugins/dist/backstage-community-plugin-nexus-repository-manager
        disabled: false
```

## 5.2. 配置

1. 在 **app-config.yaml** 文件中将代理设置为所需的 Nexus Repository Manager 服务器，如下所示：

```
proxy:
    '/nexus-repository-manager':
    target: 'https://<NEXUS_REPOSITORY_MANAGER_URL>'
    headers:
        X-Requested-With: 'XMLHttpRequest'
        # Uncomment the following line to access a private Nexus Repository Manager using a
token
        # Authorization: 'Bearer <YOUR TOKEN>'
    changeOrigin: true
    # Change to "false" in case of using self hosted Nexus Repository Manager instance with a
self-signed certificate
    secure: true
```

2. 可选：更改 Nexus 存储库管理器代理的基本 URL，如下所示：

```
nexusRepositoryManager:
  # default path is `/nexus-repository-manager`
  proxyPath: /custom-path
```

3. 可选：启用以下实验性注解：

```
nexusRepositoryManager:
  experimentalAnnotations: true
```

4. 使用以下注解注解您的实体：

```
metadata:
  annotations:
  # insert the chosen annotations here
  # example
  nexus-repository-manager/docker.image-name: `<ORGANIZATION>/<REPOSITORY>`,
```

# 第 6 章 安装和配置 TEKTON 插件

您可以使用 Tekton 插件来视觉化在 Kubernetes 或 OpenShift 集群中运行的 CI/CD 管道的结果。通过插件，用户可以视觉化查看管道中所有相关任务的高级别状态。

## 6.1. 安装

**先决条件**

- 已安装并配置了 **@backstage/plugin-kubernetes** 和 **@backstage/plugin-kubernetes-backend** 动态插件。
- 您已将 Kubernetes 插件配置为使用 **ServiceAccount** 连接到集群。
- **必须为** 自定义资源(PipelineRuns 和 TaskRuns)授予访问集群的 **ServiceAccount**。

**注意**

如果您配置了 RHDH Kubernetes 插件，则已授予 **ClusterRole**。

- 要查看 pod 日志，您被授予了 **pod/log** 的权限。
- 您可以使用以下代码为自定义资源和 pod 日志授予 **ClusterRole** ：

```
kubernetes:
  ...
  customResources:
    - group: 'tekton.dev'
      apiVersion: 'v1'
      plural: 'pipelineruns'
    - group: 'tekton.dev'
      apiVersion: 'v1'


...
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: backstage-read-only
rules:
  - apiGroups:
      - ""
    resources:
      - pods/log
    verbs:
      - get
      - list
      - watch
  ...
  - apiGroups:
      - tekton.dev
    resources:
      - pipelineruns
      - taskruns
```

```
    verbs:
      - get
      - list
```

您可以将准备的清单用于只读 **ClusterRole**，它为 Kubernetes 插件和 Tekton 插件提供访问权限。

- 在实体的 **catalog-info.yaml** 文件中添加以下注解，以识别实体是否包含 Kubernetes 资源：

```
annotations:
  ...

  backstage.io/kubernetes-id: <BACKSTAGE_ENTITY_NAME>
```

- 您还可以添加 **backstage.io/kubernetes-namespace** 注解，以使用定义的命名空间来识别 Kubernetes 资源。

```
annotations:
  ...

  backstage.io/kubernetes-namespace: <RESOURCE_NS>
```

- 将以下注解添加到实体的 **catalog-info.yaml** 文件中，以启用 RHDH 中的 Tekton 相关功能。注解的值标识 RHDH 实体的名称：

```
annotations:
  ...

  janus-idp.io/tekton : <BACKSTAGE_ENTITY_NAME>
```

- 添加自定义标签选择器，它 RHDH 用于查找 Kubernetes 资源。标签选择器优先于 ID 注解。

```
annotations:
  ...

  backstage.io/kubernetes-label-selector: 'app=my-app,component=front-end'
```

- 在资源中添加以下标签，以便 Kubernetes 插件从请求的实体获取 Kubernetes 资源：

```
labels:
  ...

  backstage.io/kubernetes-id: <BACKSTAGE_ENTITY_NAME>
```

**注意**

使用标签选择器时，上述标签必须存在于资源上。

**流程**

- Tekton 插件在带有基本配置属性的 RHDH 中预加载。要启用它，请将 disabled 属性设置为 false，如下所示：

```
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: ./dynamic-plugins/dist/backstage-community-plugin-tekton
        disabled: false
```

# 第 7 章 安装 TOPOLOGY 插件

## 7.1. 安装

Topology 插件允许您视觉化工作负载，如 Deployment、Job、Daemonset、Statefulset、CronJob、Pod 和 Virtual Machines，支持 Kubernetes 集群上的任何服务。

**先决条件**

- 已安装并配置了 @backstage/plugin-kubernetes-backend 动态插件。
- 您已将 Kubernetes 插件配置为使用 ServiceAccount 连接到集群。
- **ClusterRole** 必须被授予到集群的 ServiceAccount 访问。

**注意**

如果您配置了 Developer Hub Kubernetes 插件，则已授予 **ClusterRole**。

**流程**

- Topology 插件在 Developer Hub 中预加载，具有基本配置属性。要启用它，请将 disabled 属性设置为 false，如下所示：

  **app-config.yaml 片段**

```
auth:
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: ./dynamic-plugins/dist/backstage-community-plugin-topology
        disabled: false
```

## 7.2. 配置 TOPOLOGY 插件

### 7.2.1. 查看 OpenShift 路由

**流程**

1. 要查看 OpenShift 路由，请授予集群角色中路由资源的读取访问权限：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: backstage-read-only
rules:
  ...
  - apiGroups:
      - route.openshift.io
    resources:
```

```
      - routes
    verbs:
      - get
      - list
```

2. 另外，在 **app-config.yaml** 文件中的 **kubernetes.customResources** 属性中添加以下内容：

```
kubernetes:
  ...
  customResources:
    - group: 'route.openshift.io'
      apiVersion: 'v1'
        plural: 'routes'
```

### 7.2.2. 查看 pod 日志

**流程**

- 要查看 pod 日志，您必须为 **ClusterRole** 授予以下权限：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: backstage-read-only
rules:
  ...
  - apiGroups:
      - ''
    resources:
      - pods
      - pods/log
    verbs:
      - get
      - list
      - watch
```

### 7.2.3. 查看 Tekton PipelineRuns

**流程**

1. 要查看 Tekton PipelineRuns，请授予 **ClusterRole** 中的 **管道**、**pipelinesruns** 和 **taskruns** 资源的读取访问权限：

```
...
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: backstage-read-only
rules:
  ...
  - apiGroups:
      - tekton.dev
    resources:
```

```
      - pipelines
      - pipelineruns
      - taskruns
    verbs:
      - get
      - list
```

2. 要在侧面面板中查看 Tekton PipelineRuns 列表，以及 Topology 节点 decorator 中的最新的 PipelineRuns 状态，请在 **app-config.yaml** 文件中的 **kubernetes.customResources** 属性中添加以下代码：

```
kubernetes:
  ...
  customResources:
    - group: 'tekton.dev'
      apiVersion: 'v1'
      plural: 'pipelines'
    - group: 'tekton.dev'
      apiVersion: 'v1'
      plural: 'pipelineruns'
    - group: 'tekton.dev'
      apiVersion: 'v1'
      plural: 'taskruns'
```

### 7.2.4. 查看虚拟机

**先决条件**

1. OpenShift Virtualization Operator 在 Kubernetes 集群上安装和配置。.Procedure

2. 授予 **ClusterRole** 中 **VirtualMachines** 资源的读取访问权限：

```
...
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: backstage-read-only
rules:
  ...
  - apiGroups:
      - kubevirt.io
    resources:
      - virtualmachines
      - virtualmachineinstances
    verbs:
      - get
      - list
```

3. 要查看拓扑插件上的虚拟机节点，请将以下代码添加到 **app-config.yaml** 文件中的 **kubernetes.customResources** 属性中：

```
kubernetes:
  ...
  customResources:
```

```
    - group: 'kubevirt.io'
      apiVersion: 'v1'
      plural: 'virtualmachines'
    - group: 'kubevirt.io'
      apiVersion: 'v1'
      plural: 'virtualmachineinstances'
```

### 7.2.5. 启用源代码编辑器

要启用源代码编辑器，您必须授予对 **ClusterRole** 中的 CheClusters 资源的读取访问权限，如下例所示：

```
...
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: backstage-read-only
rules:
  ...
  - apiGroups:
      - org.eclipse.che
    resources:
      - checlusters
    verbs:
      - get
      - list
```

要使用源代码编辑器，您必须将以下配置添加到 **app-config.yaml** 文件中的 **kubernetes.customResources** 属性中：

```
kubernetes:
  ...
  customResources:
    - group: 'org.eclipse.che'
      apiVersion: 'v2'
      plural: 'checlusters'
```

## 7.3. 为 TOPOLOGY 插件管理标签和注解

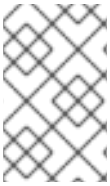
### 7.3.1. 连接到源代码编辑器或源

将以下注解添加到工作负载资源中，如 Deployments，以使用源代码编辑器导航到关联应用程序的 Git 存储库：

```
annotations:
  app.openshift.io/vcs-uri: <GIT_REPO_URL>
```

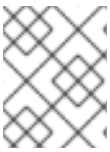
添加以下注解以导航到特定分支：

```
annotations:
  app.openshift.io/vcs-ref: <GIT_REPO_BRANCH>
```



**注意**

如果已安装并配置了 Red Hat OpenShift Dev Spaces，并且 Git URL 注解也会添加到工作负载 YAML 文件中，点编辑代码 decorator 将您重定向到 Red Hat OpenShift Dev Spaces 实例。



**注意**

当您使用 OCP Git 导入流部署应用时，您不需要添加标签，因为导入流会这样做。否则，您需要手动将标签添加到工作负载 YAML 文件中。

您还可以使用 decorator 使用您要访问的编辑 URL 添加 **app.openshift.io/edit-url** 注解。

### 7.3.2. 实体注解/标签

要使 RHDH 检测实体是否具有 Kubernetes 组件，请在实体的 **catalog-info.yaml** 文件中添加以下注解：

```
annotations:
  backstage.io/kubernetes-id: <BACKSTAGE_ENTITY_NAME>
```

在资源中添加以下标签，以便 Kubernetes 插件从请求的实体获取 Kubernetes 资源：

```
labels:
  backstage.io/kubernetes-id: <BACKSTAGE_ENTITY_NAME>`
```

**注意**

在使用标签选择器时，上述标签必须存在于资源上。

### 7.3.3. 命名空间注解

**流程**

- 要使用定义的命名空间识别 Kubernetes 资源，请添加 **backstage.io/kubernetes-namespace** 注解：

  ```
  annotations:
    backstage.io/kubernetes-namespace: <RESOURCE_NS>
  ```

  如果将 **backstage.io/kubernetes-namespace** 注解添加到 **catalog-info.yaml** 文件中，则无法使用源代码编辑器访问 Red Hat OpenShift Dev Spaces 实例。

  要检索实例 URL，您需要 CheCluster 自定义资源(CR)。因为 CheCluster CR 在 openshift-devspaces 命名空间中创建，因此如果命名空间注解值不是 openshift-devspaces，则不会检索实例 URL。

### 7.3.4. 标签选择器查询注解

您可以自行编写自定义标签，RHDH 用来查找 Kubernetes 资源。标签选择器优先于 ID 注解：

```
annotations:
  backstage.io/kubernetes-label-selector: 'app=my-app,component=front-end'
```

如果您在配置 Red Hat Dev Spaces 时有多个实体，并希望多个实体支持重定向到 Red Hat Dev Spaces 实例的编辑代码 decorator，您可以将 backstage.io/kubernetes-label-selector 注解添加到每个实体的 catalog-info.yaml 文件中。

```
annotations:
  backstage.io/kubernetes-label-selector: 'component in (<BACKSTAGE_ENTITY_NAME>,che)'
```

如果使用前面的标签选择器，您必须将以下标签添加到资源中，以便 Kubernetes 插件从请求的实体获取 Kubernetes 资源：

```
labels:
  component: che # add this label to your che cluster instance
labels:
  component: <BACKSTAGE_ENTITY_NAME> # add this label to the other resources associated with your entity
```

您还可以为带有唯一标签的标签选择器编写您自己的自定义查询，以区分您的实体。但是，您需要确保将这些标签添加到与包括 CheCluster 实例在内的实体关联的资源中。

### 7.3.5. 节点中显示的图标

要在拓扑节点中显示运行时图标，请在工作负载资源中添加以下标签，如 Deployment：

```
labels:
  app.openshift.io/runtime: <RUNTIME_NAME>
```

另外，您可以包含以下标签来显示运行时图标：

```
labels:
  app.kubernetes.io/name: <RUNTIME_NAME>
```

< **RUNTIME_NAME>** 支持的值包括：

- Django
- dotnet
- drupal
- go-gopher
- golang
- Grails
- jboss
- jruby
- js
- nginx
- nodejs

- openjdk
- perl
- phalcon
- php
- python
- Quarkus
- Rails
- redis
- rh-spring-boot
- rust
- java
- rh-openjdk
- ruby
- Spring
- spring-boot

**注意**

其他值会导致节点无法呈现图标。

### 7.3.6. 应用程序分组

要显示可视组中部署或 pod 等工作负载资源，请添加以下标签：

```
labels:
  app.kubernetes.io/part-of: <GROUP_NAME>
```

### 7.3.7. 节点连接器

**流程**

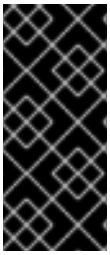
要显示使用可视连接器的部署或 pod 等工作负载资源，请添加以下注解：

+

```
annotations:
  app.openshift.io/connects-to: '[{"apiVersion": <RESOURCE_APIVERSION>,"kind":
<RESOURCE_KIND>,"name": <RESOURCE_NAME>}]'
```

如需有关标签和注解的更多信息，*请参阅 OpenShift 应用的标签和注解指南*。

# 第 8 章 批量导入 GITHUB 仓库



**重要**

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

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

Red Hat Developer Hub 可以自动加入 GitHub 存储库并跟踪它们的导入状态。

## 8.1. 启用并授予 BULK IMPORT 功能的访问权限

您可以为用户启用 Bulk Import 功能，并授予他们访问它所需的权限。

**先决条件**

- 您已配置了 GitHub 集成。

**流程**

1. Bulk Import 插件已安装，但默认禁用。要启用 **./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import-backend-dynamic** 和 **./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import** 插件，请使用以下内容编辑 **dynamic-plugins.yaml** ：

   **dynamic-plugins.yaml 片段**

   ```
   plugins:
     - package: ./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import-backend-dynamic
       disabled: false
     - package: ./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import
       disabled: false
   ```

   请参阅 Red Hat Developer Hub 中安装和查看插件 。

2. 为不是管理员的用户配置所需的 **bulk.import** RBAC 权限，如下所示：

   **rbac-policy.csv fragment**

   ```
   p, role:default/bulk-import, bulk.import, use, allow
   g, user:default/<your_user>, role:default/bulk-import
   ```

   请注意，只有 Developer Hub 管理员或具有 **bulk.import** 权限的用户才能使用 Bulk Import 功能。请参阅 Red Hat Developer Hub 中的权限策略 。

**验证**

- 侧边栏会显示一个 **Bulk Import** 选项。

- Bulk Import 页面显示 Added Repositories 列表。

## 8.2. 导入多个 GITHUB 仓库

在 Red Hat Developer Hub 中，您可以选择 GitHub 存储库，并将其载入到 Developer Hub 目录。

**先决条件**

- 您已 启用了 Bulk Import 功能，并可以访问它。

**流程**

1. 单击左侧栏中的 **Bulk Import**。
2. 点右上角的 **Add** 按钮，查看从配置的 GitHub 集成访问的所有存储库的列表。
   a. 从 **Repositories** 视图中，您可以选择任何存储库，或搜索任何可访问的存储库。对于每个选择的软件仓库，都会生成一个 **catalog-info.yaml**。
   b. 在 **Organizations** 视图中，您可以通过单击第三列中的 **Select** 来选择任何机构。这个选项允许您从所选机构中选择一个或多个软件仓库。
3. 点 **Preview file** 查看或编辑每个存储库的拉取请求详情。
   a. 查看拉取请求描述和 **catalog-info.yaml** 文件内容。
   b. 可选：当仓库有 **.github/CODEOWNERS** 文件时，您可以选择 **Use CODEOWNERS 文件作为实体 Owner** 复选框来使用它，而不是 **content-info.yaml** 包含特定的实体所有者。
   c. 点击 **Save**。
4. 单击 **Create pull requests**。此时，对所选存储库运行一组空运行检查，以确保它们满足导入的要求，例如：
   a. 使用存储库 **catalog-info.yaml**中指定的名称验证 Developer Hub 目录中没有实体
   b. 验证存储库是否不为空
   c. 如果 Use CODEOWNERS 文件为那个仓库选择了 **Entity Owner 复选框，则验证仓库是否包含 .github/CODEOWNERS 文件**
      - 如果发生错误，则不会创建拉取请求，您会看到 *Failed to create PR* 错误消息详细描述了这个问题。要查看有关原因的更多详细信息，请点 **编辑**。
      - 如果没有错误，则创建拉取请求，并将您重定向到添加的存储库列表。
5. 检查并合并创建 **catalog-info.yml** 文件的每个拉取请求。

**验证**

- **Added repositories** 列表显示您导入的存储库，每个软件仓库都有一个适当的状态： *Waiting for approval* 或 *Added*。
- 对于列出 *批准作业的每个 Waiting*，在对应的仓库中添加 **catalog-info.yaml** 文件对应的拉取请求。

## 8.3. 管理添加的软件仓库

您可以仔细阅读并管理导入到 Developer Hub 的存储库。

**先决条件**

- 您已 导入了 GitHub 存储库。

**流程**

1. 单击左侧栏中的 **Bulk Import**，以显示所有正在跟踪为 Import jobs 的存储库，以及它们的状态。

   **已添加**

   在导入拉取请求被合并后，仓库会添加到 Developer Hub 目录中，或者仓库已在批量导入过程中包含 **catalog-info.yaml** 文件。请注意，可能需要过几分钟后，实体才会在目录中可用。

   **等待批准**

   有一个打开的拉取请求，将 **catalog-info.yaml** 文件添加到存储库中。您可以：

   - 点 **右侧的铅笔图标** 查看拉取请求的详情，或编辑 Developer Hub 中的拉取请求内容。
   - 删除 Import 作业，此操作也会关闭导入 PR。
   - 要将 Import 任务转换为 *Added* 状态，请从 Git 存储库合并导入拉取请求。

   **空**

   Developer Hub 无法决定导入作业状态，因为存储库会从其他源导入，但没有 **catalog-info.yaml** 文件，且缺少任何导入拉取请求来添加它。

**注意**

- 合并导入拉取请求后，导入状态会在 *Added* Repositories 列表中被标记为 Added，但可能需要几秒钟才能出现在 Developer Hub Catalog 中。
- 如果满足以下条件，通过其他源添加的位置（例如，在 **app-config.yaml** 文件中静态添加）、在启用 GitHub 发现 时动态添加的位置，或者在满足以下条件时在 Added Repositories 的 Bulk Import 列表中手动注册：
  - 目标存储库可从配置的 GitHub 集成访问。
  - 位置 URL 指向存储库默认分支根目录下的 **catalog-info.yaml** 文件。

## 8.4. 了解 BULK IMPORT AUDIT LOGS

Bulk Import backend 插件将以下事件添加到 Developer Hub 审计日志中。有关如何配置 和查看审计日志的更多信息，请参阅 Red Hat Developer Hub 中的审计日志。

**批量导入事件 ：**

**BulkImportUnknownEndpoint**

跟踪对未知端点的请求。

**BulkImportPing**

跟踪对 **/ping** 端点的 **GET** 请求，这允许我们确保批量导入后端已启动并在运行。

**BulkImportFindAllOrganizations**

跟踪 **GET** 请求到 **/organizations** 端点，该端点返回所有配置的 GitHub 集成可访问的机构列表。

**BulkImportFindRepositoriesByOrganization**

跟踪对 **/organizations/:orgName/repositories** 端点的 **GET** 请求，该端点返回指定机构的存储库列表（可以从任何配置的 GitHub 集成访问）。

**BulkImportFindAllRepositories**

跟踪 GET 请求到 **/repositories** 端点，该端点返回可从所有配置的 GitHub 集成访问的存储库列表。

**BulkImportFindAllImports**

跟踪 **GET** 请求到 **/imports** 端点，该端点返回现有导入作业的列表及其状态。

**BulkImportCreateImportJobs**

通过最终在目标仓库中创建导入拉取请求，将 **POST** 请求跟踪到 **/imports** 端点，允许向 Developer Hub 目录提交对一个或多个存储库的请求。

**BulkImportFindImportStatusByRepo**

跟踪对 **/import/by-repo** 端点的 **GET** 请求，该端点获取指定存储库导入作业的详情。

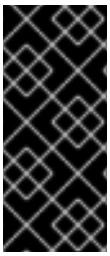
**BulkImportDeleteImportByRepo**

通过关闭任何可以创建的开放导入拉取请求，将跟踪对 **/import/by-repo** 端点的 **DELETE** 请求，该端点会删除指定存储库的任何现有导入作业。

### 批量导入审计日志示例


```
{
  "actor": {
    "actorId": "user:default/myuser",
    "hostname": "localhost",
    "ip": "::1",
    "userAgent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/128.0.0.0 Safari/537.36"
  },
  "eventName": "BulkImportFindAllOrganizations",
  "isAuditLog": true,
  "level": "info",
  "message": "'get /organizations' endpoint hit by user:default/myuser",
  "meta": {},
  "plugin": "bulk-import",
  "request": {
    "body": {},
    "method": "GET",
    "params": {},
    "query": {
      "pagePerIntegration": "1",
      "sizePerIntegration": "5"
    },
    "url": "/api/bulk-import/organizations?pagePerIntegration=1&sizePerIntegration=5"
  },
  "response": {
    "status": 200
  },
  "service": "backstage",
  "stage": "completion",
  "status": "succeeded",
  "timestamp": "2024-08-26 16:41:02"
}
```

# 第 9 章 RED HAT DEVELOPER HUB 中的 SERVICENOW 自定义操作



**重要**

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

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

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

Developer Hub 中的自定义操作允许您促进和自动管理记录。使用自定义操作，您可以执行以下操作：

- 创建、更新或删除记录
- 检索有关单个记录或多个记录的信息

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

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

**先决条件**

- Red Hat Developer Hub 已安装并运行。有关安装 Developer Hub 的更多信息，请参阅使用 Helm Chart 在 OpenShift Container Platform 上安装 Red Hat Developer Hub 。
- 您已在 Developer Hub 中创建了一个项目。

**流程**

1. 要激活自定义操作插件，请添加带有 **插件名称的软件包**，并更新 Helm Chart 中的 **disabled** 字段，如下所示：

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

**注意**

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

2. 在 Helm Chart 中设置以下变量来访问自定义操作：

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

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

ServiceNow 自定义操作允许您管理 Red Hat Developer Hub 中的记录。自定义操作支持以下 API 请求的 HTTP 方法：

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

### 9.2.1. ServiceNow 自定义操作

**[GET] servicenow:now:table:retrieveRecord**

从 Developer Hub 中的表检索指定记录的信息。

**表 9.1. 输入参数**

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

| Name | 类型 | 要求 | 描述 |
| --- | --- | --- | --- |
| **sysparmQueryNoDomain** | **布尔值** | 选填 | 将 设置为 **true**，以在域间访问数据（如果授权）。默认值为 **false**。 |

**表 9.2. 输出参数**

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

[GET] servicenow:now:table:retrieveRecords

从 Developer Hub 中的表检索有关多个记录的信息。

**表 9.3. 输入参数**

| Name | 类型 | 要求 | 描述 |
| --- | --- | --- | --- |
| **tableName** | **string** | 必填 | 从中检索记录的表的名称 |
| **sysparamQuery** | **string** | 选填 | 用于过滤结果的编码查询字符串 |
| **sysparmDisplayValue** | **enum("true", "false", "all")** | 选填 | 返回字段显示值，如 **true**，实际值为 **false**，或两者。默认值为 **false**。 |
| **sysparmExcludeReferenceLink** | **布尔值** | 选填 | 设置为 **true** 以排除参考字段的 Table API 链接。默认值为 **false**。 |
| **sysparmSuppressPaginationHeader** | **布尔值** | 选填 | 设置为 **true** 以阻止分页标头。默认值为 **false**。 |
| **sysparmFields** | **string[]** | 选填 | 要在响应中返回的字段数组 |
| **sysparmLimit** | **int** | 选填 | 每个页面返回的最大结果数。默认值为 **10,000**。 |
| **sysparmView** | **string** | 选填 | 根据指定的 UI 视图呈现响应。您可以使用 **sysparm_fields** 覆盖此参数。 |

| Name | 类型 | 要求 | 描述 |
| --- | --- | --- | --- |
| **sysparmQueryCategory** | **string** | 选填 | 用于查询的查询类别的名称 |
| **sysparmQueryNoDomain** | **布尔值** | 选填 | 将 设置为 **true**，以在域间访问数据（如果授权）。默认值为 **false**。 |
| **sysparmNoCount** | **布尔值** | 选填 | 不要在表中执行所选 countrolebinding。默认值为 **false**。 |

**表 9.4. 输出参数**

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

[POST] servicenow:now:table:createRecord

在 Developer Hub 的表中创建记录。

**表 9.5. 输入参数**

| Name | 类型 | 要求 | 描述 |
| --- | --- | --- | --- |
| **tableName** | **string** | 必填 | 保存记录的表的名称 |
| **requestBody** | **record<PropertyKey, unknown>** | 选填 | 在指定记录中定义的每个参数的字段名称和关联值 |
| **sysparmDisplayValue** | **enum("true", "false", "all")** | 选填 | 返回字段显示值，如 **true**，实际值为 **false**，或两者。默认值为 **false**。 |
| **sysparmExcludeReferenceLink** | **布尔值** | 选填 | 设置为 **true** 以排除参考字段的 Table API 链接。默认值为 **false**。 |
| **sysparmFields** | **string[]** | 选填 | 要在响应中返回的字段数组 |

| Name | 类型 | 要求 | 描述 |
| --- | --- | --- | --- |
| **sysparmInputDisplayValue** | **布尔值** | 选填 | 使用其显示值（如 **true** 或实际值）设置字段值，如 **false**。默认值为 **false**。 |
| **sysparmSuppressAutoSysField** | **布尔值** | 选填 | 设置为 **true** 以禁止自动生成系统字段。默认值为 **false**。 |
| **sysparmView** | **string** | 选填 | 根据指定的 UI 视图呈现响应。您可以使用 **sysparm_fields** 覆盖此参数。 |

**表 9.6. 输出参数**

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

[PUT] servicenow:now:table:modifyRecord

修改 Developer Hub 中的表中的记录。

**表 9.7. 输入参数**

| Name | 类型 | 要求 | 描述 |
| --- | --- | --- | --- |
| **tableName** | **string** | 必填 | 修改记录的表的名称 |
| **sysId** | **string** | 必填 | 要修改的记录的唯一标识符 |
| **requestBody** | **record<PropertyKey, unknown>** | 选填 | 在指定记录中定义的每个参数的字段名称和关联值 |
| **sysparmDisplayValue** | **enum("true", "false", "all")** | 选填 | 返回字段显示值，如 **true**，实际值为 **false**，或两者。默认值为 **false**。 |
| **sysparmExcludeReferenceLink** | **布尔值** | 选填 | 设置为 **true** 以排除参考字段的 Table API 链接。默认值为 **false**。 |
| **sysparmFields** | **string[]** | 选填 | 要在响应中返回的字段数组 |
| **sysparmInputDisplayValue** | **布尔值** | 选填 | 使用其显示值（如 **true** 或实际值）设置字段值，如 **false**。默认值为 **false**。 |

| Name | 类型 | 要求 | 描述 |
| --- | --- | --- | --- |
| **sysparmSuppressAutoSysField** | **布尔值** | 选填 | 设置为 **true** 以禁止自动生成系统字段。默认值为 **false**。 |
| **sysparmView** | **string** | 选填 | 根据指定的 UI 视图呈现响应。您可以使用 **sysparm_fields** 覆盖此参数。 |
| **sysparmQueryNoDomain** | **布尔值** | 选填 | 将 设置为 **true**，以在域间访问数据（如果授权）。默认值为 **false**。 |

**表 9.8. 输出参数**

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

[PATCH] servicenow:now:table:updateRecord

更新 Developer Hub 中的表中的记录。

**表 9.9. 输入参数**

| Name | 类型 | 要求 | 描述 |
| --- | --- | --- | --- |
| **tableName** | **string** | 必填 | 更新记录的表的名称 |
| **sysId** | **string** | 必填 | 要更新记录的唯一标识符 |
| **requestBody** | **record<PropertyKey, unknown>** | 选填 | 在指定记录中定义的每个参数的字段名称和关联值 |
| **sysparmDisplayValue** | **enum("true", "false", "all")** | 选填 | 返回字段显示值，如 **true**，实际值为 **false**，或两者。默认值为 **false**。 |
| **sysparmExcludeReferenceLink** | **布尔值** | 选填 | 设置为 **true** 以排除参考字段的 Table API 链接。默认值为 **false**。 |
| **sysparmFields** | **string[]** | 选填 | 要在响应中返回的字段数组 |

| Name | 类型 | 要求 | 描述 |
| --- | --- | --- | --- |
| **sysparmInputDisplayValue** | **布尔值** | 选填 | 使用其显示值（如 **true** 或实际值）设置字段值，如 **false**。默认值为 **false**。 |
| **sysparmSuppressAutoSysField** | **布尔值** | 选填 | 设置为 **true** 以禁止自动生成系统字段。默认值为 **false**。 |
| **sysparmView** | **string** | 选填 | 根据指定的 UI 视图呈现响应。您可以使用 **sysparm_fields** 覆盖此参数。 |
| **sysparmQueryNoDomain** | **布尔值** | 选填 | 将 设置为 **true**，以在域间访问数据（如果授权）。默认值为 **false**。 |

**表 9.10. 输出参数**

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

[DELETE] servicenow:now:table:deleteRecord

从 Developer Hub 中的表中删除记录。

**表 9.11. 输入参数**

| Name | 类型 | 要求 | 描述 |
| --- | --- | --- | --- |
| **tableName** | **string** | 必填 | 从中删除记录的表的名称 |
| **sysId** | **string** | 必填 | 要删除记录的唯一标识符 |
| **sysparmQueryNoDomain** | **布尔值** | 选填 | 将 设置为 **true**，以在域间访问数据（如果授权）。默认值为 **false**。 |

# 第 10 章 RED HAT DEVELOPER HUB 中的 KUBERNETES 自定义操作

**重要**

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

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

使用 Kubernetes 自定义操作，您可以创建和管理 Kubernetes 资源。

默认情况下，在 Developer Hub 实例上预装并禁用了 Kubernetes 自定义操作插件。您可以通过配置 Red Hat Developer Hub Helm Chart 来禁用或启用 Kubernetes 自定义操作插件，并更改其他参数。

**注意**

Kubernetes 构建程序操作和 Kubernetes 自定义操作在整个文档中引用同样的概念。

## 10.1. 在 RED HAT DEVELOPER HUB 中启用 KUBERNETES 自定义操作插件

在 Red Hat Developer Hub 中，Kubernetes 自定义操作作为预安装插件提供，该插件默认是禁用的。您可以通过更新 Helm chart 中的 **禁用** 键值来启用 Kubernetes 自定义操作插件。

**先决条件**

- 已使用 Helm Chart 安装 Red Hat Developer Hub。

**流程**

要启用 Kubernetes 自定义操作插件，请完成以下步骤：

- 在 Helm Chart 中，使用 Kubernetes 自定义操作插件名称添加软件包并更新 **disabled** 字段。**例如：**

```
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: ./dynamic-plugins/dist/backstage-community-plugin-scaffolder-backend-module-kubernetes-dynamic
        disabled: false
```

**注意**

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

## 10.2. 在 RED HAT DEVELOPER HUB 中使用 KUBERNETES 自定义操作插件

在 Red Hat Developer Hub 中，Kubernetes 自定义操作允许您为 Kubernetes 运行模板操作。

**流程**

- 要在自定义模板中使用 Kubernetes 自定义操作，请在模板中添加以下 Kubernetes 操作：

```
action: kubernetes:create-namespace
id: create-kubernetes-namespace
name: Create kubernetes namespace
input:
  namespace: my-rhdh-project
  clusterRef: bar
  token: TOKEN
  skipTLSVerify: false
  caData: Zm9v
  labels: app.io/type=ns; app.io/managed-by=org;
```

**其他资源**

- 配置模板.

## 10.3. 使用 RED HAT DEVELOPER HUB 中的 KUBERNETES 自定义操作创建模板

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。它还包含所需的输入变量，以及构建服务所执行的操作列表。

+

```
apiVersion: scaffolder.backstage.io/v1beta3
kind: Template
metadata:
  name: create-kubernetes-namespace
  title: Create a kubernetes namespace
  description: Create a kubernetes namespace

spec:
  type: service
  parameters:
    - title: Information
```

```
    required: [namespace, token]
    properties:
      namespace:
        title: Namespace name
        type: string
        description: Name of the namespace to be created
      clusterRef:
        title: Cluster reference
        type: string
        description: Cluster resource entity reference from the catalog
        ui:field: EntityPicker
        ui:options:
          catalogFilter:
            kind: Resource
      url:
        title: Url
        type: string
        description: Url of the kubernetes API, will be used if clusterRef is not provided
      token:
        title: Token
        type: string
        ui:field: Secret
        description: Bearer token to authenticate with
      skipTLSVerify:
        title: Skip TLS verification
        type: boolean
        description: Skip TLS certificate verification, not recommended to use in production
environment, default to false
      caData:
        title: CA data
        type: string
        ui:field: Secret
        description: Certificate Authority base64 encoded certificate
      labels:
        title: Labels
        type: string
        description: Labels to be applied to the namespace
        ui:widget: textarea
        ui:options:
          rows: 3
        ui:help: 'Hint: Separate multiple labels with a semicolon!'
        ui:placeholder: 'kubernetes.io/type=namespace; app.io/managed-by=org'

  steps:
    - id: create-kubernetes-namespace
      name: Create kubernetes namespace
      action: kubernetes:create-namespace
      input:
        namespace: ${{ parameters.namespace }}
        clusterRef: ${{ parameters.clusterRef }}
        url: ${{ parameters.url }}
        token: ${{ secrets.token }}
        skipTLSVerify: ${{ parameters.skipTLSVerify }}
        caData: ${{ secrets.caData }}
        labels: ${{ parameters.labels }}
```

### 10.3.1. Red Hat Developer Hub 中支持的 Kubernetes 自定义操作

在 Red Hat Developer Hub 中，您可以在 scaffolder 模板中使用自定义 Kubernetes 操作。

**自定义 Kubernetes 构建程序操作**

**action: kubernetes:create-namespace**

在 Developer Hub 中为 Kubernetes 集群创建一个命名空间。

| 参数名称 | 类型 | 要求 | 描述 | Example |
|---|---|---|---|---|
| **namespace** | **string** | 必填 | Kubernetes 命名空间的名称 | **my-rhdh-project** |
| **clusterRef** | **string** | 仅在未定义 **url** 时才需要。您不能指定 **url** 和 **clusterRef**。 | 来自目录的集群资源实体引用 | **bar** |
| **url** | **string** | 仅在未定义 **clusterRef** 时才需要。您不能指定 **url** 和 **clusterRef**。 | Kubernetes 集群的 API url | **https://api.foo.redhat.com:6443** |
| **token** | **字符串** | 必填 | 用于身份验证的 Kubernetes API bearer 令牌 | |
| **skipTLSVerify** | **布尔值** | 选填 | 如果为 true，则会跳过证书验证 | false |
| **caData** | **string** | 选填 | Base64 编码的证书数据 | |
| **label** | **string** | 选填 | 应用到命名空间的标签 | app.io/type=ns; app.io/managed-by=org; |

# 第 11 章 覆盖核心后端服务配置

Red Hat Developer Hub (RHDH)后端平台由多个已封装的核心服务组成。RHDH 后端在初始化过程中静态安装这些默认核心服务。

您可以通过自定义后端源代码和重建 Developer Hub 应用程序来配置这些核心服务。或者，您可以使用动态插件功能将核心服务安装为 BackendFeature 来自定义核心服务。

要使用动态插件功能在 RHDH 应用程序中自定义核心服务，您必须配置后端以避免静态安装给定的默认核心服务。

例如，可以通过为 root HTTP 路由器后端服务安装自定义配置功能来处理所有传入请求，从而添加中间件功能来处理所有传入的请求，以允许访问底层 Express 应用。

处理传入 HTTP 请求的 BackendFeature 中间件功能示例

```
// Create the BackendFeature
export const customRootHttpServerFactory: BackendFeature =
  rootHttpRouterServiceFactory({
    configure: ({ app, routes, middleware, logger }) => {
      logger.info(
        'Using custom root HttpRouterServiceFactory configure function',
      );
      app.use(middleware.helmet());
      app.use(middleware.cors());
      app.use(middleware.compression());
      app.use(middleware.logging());
      // Add a the custom middleware function before all
      // of the route handlers
      app.use(addTestHeaderMiddleware({ logger }));
      app.use(routes);
      app.use(middleware.notFound());
      app.use(middleware.error());
    },
  });

// Export the BackendFeature as the default entrypoint
export default customRootHttpServerFactory;
```

在上例中，因为 BackendFeature 覆盖 HTTP 路由器服务的默认实现，您必须将

ENABLE_CORE_ROOTHTTPROUTER_OVERRIDE 环境变量设置为 true，以便 Developer Hub 不会自动安装默认实施。

### 11.1. 覆盖环境变量

要允许动态插件加载核心服务覆盖，您必须启动 Developer Hub 后端，并将对应的核心服务 ID 环境变量设置为 true。

表 11.1. 环境变量和核心服务 ID

| 变量 | 描述 |
| --- | --- |
| ENABLE_CORE_AUTH_OVERRIDE | 覆盖 core.auth 服务 |
| ENABLE_CORE_CACHE_OVERRIDE | 覆盖 core.cache 服务 |
| ENABLE_CORE_ROOTCONFIG_OVERRIDE | 覆盖 core.rootConfig 服务 |
| ENABLE_CORE_DATABASE_OVERRIDE | 覆盖 core.database 服务 |
| ENABLE_CORE_DISCOVERY_OVERRIDE | 覆盖 core.discovery 服务 |
| ENABLE_CORE_HTTPAUTH_OVERRIDE | 覆盖 core.httpAuth 服务 |
| ENABLE_CORE_HTTPROUTER_OVERRIDE | 覆盖 core.httpRouter 服务 |
| ENABLE_CORE_LIFECYCLE_OVERRIDE | 覆盖 core.lifecycle 服务 |
| ENABLE_CORE_LOGGER_OVERRIDE | 覆盖 core.logger 服务 |
| ENABLE_CORE_PERMISSIONS_OVERRIDE | 覆盖 core.permissions 服务 |
| ENABLE_CORE_ROOTHEALTH_OVERRIDE | 覆盖 core.rootHealth 服务 |
| ENABLE_CORE_ROOTHTTPROUTER_OVERRIDE | 覆盖 core.rootHttpRouter 服务 |
| ENABLE_CORE_ROOTLIFECYCLE_OVERRIDE | 覆盖 core.rootLifecycle 服务 |
| ENABLE_CORE_SCHEDULER_OVERRIDE | 覆盖 core.scheduler 服务 |
| ENABLE_CORE_USERINFO_OVERRIDE | 覆盖 core.userInfo 服务 |
| ENABLE_CORE_URLREADER_OVERRIDE | 覆盖 core.urlReader 服务 |

| 变量 | 描述 |
| --- | --- |
| **ENABLE_EVENTS_SERVICE_OVERRIDE** | 覆盖 **events.service** 服务 |