OpenShift Container Platform 4.16

Web 控制台

在 OpenShift Container Platform 中使用 web 控制台

Last Updated: 2024-07-02

OpenShift Container Platform 4.16 Web 控制台

在 OpenShift Container Platform 中使用 web 控制台

法律通告

摘要

本文档提供了有关使用和定制 OpenShift Container Platform web 控制台的信息。

目录

第 1 章 WEB 控制台概述

Red Hat OpenShift Container Platform Web 控制台提供了一个图形用户界面，用于视觉化您的项目数据，并执行管理、管理和故障排除任务。Web 控制台在 openshift-console 项目的 control plane 节点上运行。它由一个 **console-operator** pod 管理。支持**管理**员和**开发者**视角。

管理员和**开发者**视角都允许您为 OpenShift Container Platform 创建快速开始指南。快速开始是关于用户任务的指导教程，可用于熟悉应用程序、Operator 或其他产品。

1.1. 关于 WEB 控制台中的 ADMINISTRATOR 视角

Administrator 视角可让您查看集群清单、容量、常规和特定使用信息以及重要事件的流，它们可帮助您简化计划和故障排除任务。项目管理员和开发人员可以使用**管理**员视角。

集群管理员还可在 OpenShift Container Platform 4.7 及之后的版本中为 web 终端 Operator 打开内嵌的命令行终端实例。

注意

显示的默认 Web 控制台视角取决于用户的角色。如果用户被视为管理员，则默认显示**管理**员视角。

管理员视角提供特定于管理员用例的工作流，例如：

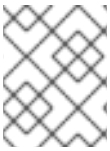
- 管理工作负载、存储、网络和集群设置。
- 使用 Operator Hub 安装和管理 Operator。
- 添加允许用户通过角色和角色绑定登录和管理用户访问权限的身份提供程序。
- 查看和管理各种高级设置，如集群更新、部分集群更新、集群 Operator、自定义资源定义 (CRD)、角色绑定和资源配额。
- 访问和管理监控功能，如指标、警报和监控仪表板。
- 查看并管理有关集群的日志记录、指标和高状态信息。
- 在 OpenShift Container Platform 中，以视觉化的方式和与**管理**员视角关联的应用程序、组件和服务交互。

1.2. 关于 WEB 控制台中的开发者视角

开发者视角提供了几个用来部署应用程序、服务和数据库的内置方法。在**开发者**视角中，您可以：

- 查看组件上滚动和重新创建推出部署的实时视觉化。
- 查看应用状态、资源利用率、项目事件流和配额消耗。
- 将您的项目与他人共享。
- 通过在项目上运行 Prometheus Query Language(PromQL)查询并查看图表中呈现的指标来排除应用程序的问题。此指标数据提供有关集群以及要监控的任何用户定义工作负载的状态信息。

集群管理员也可以在 OpenShift Container Platform 4.7 及之后的版本中的 Web 控制台中打开内嵌的命令行终端实例。



注意

显示的默认 Web 控制台视角取决于用户的角色。如果用户是开发人员，则 **Developer** 视角会被默认显示。

Developer 视角提供开发人员用例特有的工作流，比如：

- 通过导入现有代码基、镜像和容器文件在 OpenShift Container Platform 中创建和部署应用程序。
- 在一个项目中，以可视的形式和与其关联的应用程序、组件和服务进行交互，并监控它们的部署和构建状态。
- 在应用程序中对组件进行分组，并在应用程序内部及跨应用程序间连接组件。
- 集成无服务器功能（技术预览）。
- 使用 Eclipse Che 创建开发平台来编辑应用程序代码。

您可以使用 **Topology** 视图来显示项目的应用程序、组件和工作负载。如果项目中没有工作负载，则 **Topology** 视图将显示一些创建或导入它们的链接。您还可以使用 **Quick Search** 来直接导入组件。

其它资源

如需有关在 **Developer** 视角中使用 Topology 视图的更多信息，请参阅使用 **Topology** 视图查看应用程序组成。

1.3. 访问视角

您可以通过 web 控制台访问 **Administrator** 和 **Developer** 视角，如下所示：

先决条件

要访问视角，请确保您已登陆到 web 控制台。您的默认视角由用户权限自动决定。对有权访问所有项目的用户选择 **Administrator** 视角，而对于对自己项目具有有限访问权限的用户选择 **Developer** 视角

其它资源

有关更改视角的更多信息，请参阅添加用户首选项。https://docs.openshift.com/container-platform/4.16/web_console/adding-user-preferences.html

流程

1. 使用视角切换器切换到 **Administrator** 或 **Developer** 视角。
2. 从 **Project** 下拉列表中选择一个现有项目。您也可以从此下拉菜单创建新项目。

注意

您只能以 **cluster-admin** 用户身份使用视角切换器。

其他资源

- 了解有关集群管理员的更多信息
- **管理员**视角概述

- 使用 **Developer** 视角在 OpenShift Container Platform 中创建并部署应用程序
- 在项目中查看应用程序，验证应用程序的部署状态，并在 **Topology** 视图中与应用程序交互
- 查看集群信息
- 配置Web控制台
- 自定义 Web 控制台
- 关于 Web 控制台
- 使用 web 终端
- 创建快速启动指南
- 禁用Web控制台

第 2 章 访问WEB控制台

OpenShift Container Platform Web控制台是可从Web浏览器访问的用户界面。开发人员可以使用Web控制台来直观地浏览并管理项目的内容。

2.1. 先决条件

- 必须启用JavaScript才能使用Web控制台。为获得最佳体验，请使用支持WebSockets的Web浏览器。
- 在为集群创建支持基础结构之前，请参阅OpenShift Container Platform 4.x Tested Integrations页。

2.2. 了解和访问WEB控制台

Web 控制台作为 pod 在 control plane 节点上运行。这个 pod 提供了运行Web控制台所需的静态环境。

使用 **openshift-install create cluster** 命令安装 OpenShift Container Platform 后，您可以在安装程序的 CLI 输出中找到已安装集群的 Web 控制台 URL 和登录凭证。例如：

输出示例

```
INFO Install complete!
INFO Run 'export KUBECONFIG=<your working directory>/auth/kubeconfig' to manage the cluster with 'oc', the OpenShift CLI.
INFO The cluster is ready when 'oc login -u kubeadmin -p <provided>' succeeds (wait a few minutes).
INFO Access the OpenShift web-console here: https://console-openshift-console.apps.demo1.openshift4-beta-abcorp.com
INFO Login to the console with user: kubeadmin, password: <provided>
```

使用这些信息登录并访问Web控制台。

对于不是您安装的、已存在的集群，可以使用 **oc whoami --show-console** 查看 web 控制台 URL。

重要

dir 参数指定 **assets** 目录，它存储清单文件、ISO 镜像以及 **auth** 目录。**auth** 目录存储 **kubeadmin-password** 和 **kubeconfig** 文件。以 **kubeadmin** 用户身份，您可以使用 **kubeconfig** 文件通过以下设置访问集群： **export KUBECONFIG=<install_directory>/auth/kubeconfig**。**kubeconfig** 特定于生成的 ISO 镜像，因此如果设置了 **kubeconfig** 且 **oc** 命令失败，则系统可能无法使用生成的 ISO 镜像引导。要在 bootstrap 过程中执行调试，您可以使用 **kubeadmin-password** 文件的内容以 **core** 用户身份登录控制台。

其他资源

- 使用 Web 控制台启用功能集

第 3 章 使用 OPENSHIFT CONTAINER PLATFORM DASHBOARD 获取集群信息

OpenShift Container Platform Web 控制台会捕获集群的高级别信息。

3.1. 关于 OPENSHIFT CONTAINER PLATFORM 仪表板页

访问 OpenShift Container Platform 仪表板，它捕获了有关集群的高级别信息，方法是进入到 OpenShift Container Platform Web 控制台中的 **Home → Overview**。

OpenShift Container Platform 仪表板提供各种集群信息，被分别显示在独立的仪表板卡中。

OpenShift Container Platform 仪表板由以下各卡组成：

- **Details** 提供有关信息型集群详情的简单概述。
  状态包括 **ok**、**error**、**warning**、**in progress** 和 **unknown**。资源可添加自定义状态名称。
  - 集群 ID
  - 提供者
  - 版本
- **Cluster Inventory** 详细列出资源数目和相关状态。这在通过干预解决问题时非常有用，其中包含以下相关信息：
  - 节点数
  - pod 数量
  - 持久性存储卷声明
  - 集群中的裸机主机，根据其状态列出（只在 **metal3** 环境中可用）
- **Status** 可帮助管理员了解集群资源是如何被消耗的。点一个资源可以进入一个包括详细信息的页面，它列出了对指定集群资源（CPU 、内存或者存储）消耗最多的 Pod 和节点。
- **Cluster Utilization** 显示指定时间段内各种资源的容量，以帮助管理员了解高资源消耗的规模和频率，包括以下信息：
  - CPU 时间
  - 内存分配
  - 所消耗的存储
  - 所消耗的网络资源
  - Pod 数量
- **Activity** 列出了与集群中最近活动相关的消息，如创建 pod 或虚拟机迁移到另一台主机。

3.2. 识别资源和项目配额和配额

您可以在 web 控制台的 **Developer** 视角的 **Topology** 视图中查看可用资源的图形表示。

如果资源有一个关于达到资源限制或配额的消息，则会针对资源名称出现黄色的边框。点资源以打开侧面板来查看消息。如果 Topology 视图被缩放，则一个黄色点表示信息可用。

如果您在 View Shortcuts 菜单中使用 List View，则资源会显示为列表。Alerts 列指示消息是否可用。

第 4 章 添加用户首选项

您可以更改您的配置集的默认首选项以满足您的要求。您可以设置默认项目、拓扑视图（图形或列表）、编辑介质（信息或 YAML）、语言首选项和资源类型。

对用户首选项所做的更改会自动保存。

4.1. 设置用户首选项

您可以为集群设置默认用户首选项。

流程

1. 使用您的登录凭证登录到 OpenShift Container Platform web 控制台。
2. 使用 masthead 访问用户配置文件下的用户首选项。
3. 在 **General** 部分中：
   a. 在 **Theme** 字段中，您可以设置您要工作的主题。每次登录时，控制台默认为所选主题。
   b. 在 **Perspective** 字段中，您可以设置要登录到的默认视角。您可以根据需要选择 **Administrator** 或 **Developer** 视角。如果未选择透视图，您会登录到您最近看到的视角。
   c. 在 **Project** 字段中，选择一个您要处理的项目。每次登录时，控制台默认为项目。
   d. 在 **Topology** 字段中，您可以将拓扑视图设置为 default 到 graph 或 list 视图。如果未选中，控制台将默认为您使用的最后一个视图。
   e. 在 **Create/Edit resource method** 字段中，您可以设置创建或编辑资源的首选。如果表单和 YAML 选项都可用，控制台默认为您的选择。
4. 在 **Language** 部分中，选择 **默认浏览器语言** 以使用默认浏览器语言设置。否则，请选择您要用于控制台的语言。
5. 在 **Notifications** 部分中，您可以在 **Overview** 页面或 notification drawer 中切换为特定项目创建的显示通知。
6. 在 **Applications** 部分：
   a. 您可以查看默认**资源类型**。例如，如果安装了 OpenShift Serverless Operator，则默认资源类型为 **Serverless Deployment**。否则，默认的资源类型为 **Deployment**。
   b. 您可以从 **Resource Type** 字段选择另一个资源类型作为默认资源类型。

第 5 章 在OPENSHIFT CONTAINER PLATFORM中配置WEB控制台

您可以修改 OpenShift Container Platform Web 控制台来设置注销重定向 URL 或禁用快速启动指南。

5.1. 先决条件

- 部署一个OpenShift Container Platform集群。

5.2. 配置WEB控制台

您可以通过编辑**console.config.openshift.io**资源来配置Web控制台设置。

- 编辑**console.config.openshift.io**资源：

```
$ oc edit console.config.openshift.io cluster
```

以下是控制台的资源定义示例：

```
apiVersion: config.openshift.io/v1
kind: Console
metadata:
  name: cluster
spec:
  authentication:
    logoutRedirect: "" (1)
status:
  consoleURL: "" (2)
```

(1) 指定用户注销后，Web控制台要加载页面的URL。如果未指定，则用户将会返回到Web控制台的登录页面。通过指定**logoutRedirect** URL，用户可以使用身份供应商的单点注销（SLO）功能销毁其单点登录会话。

(2) Web控制台URL。要将它更新为自定义值，请参阅**自定义 Web 控制台 URL**。

5.3. 在 WEB 控制台中禁用快速启动

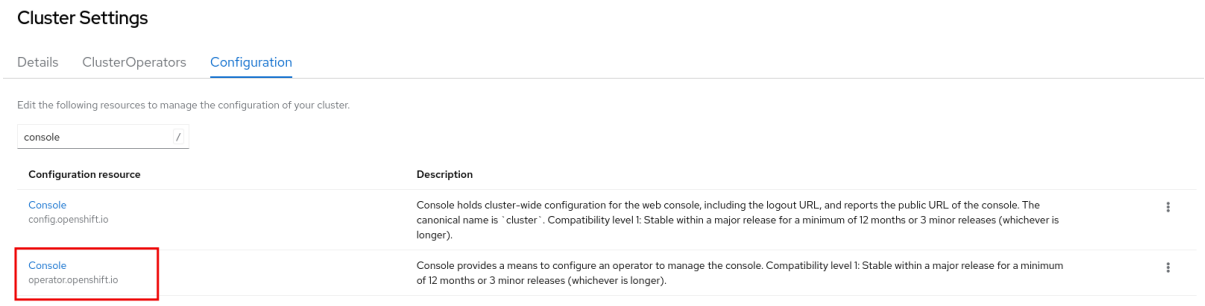
您可以使用 Web 控制台的 **Administrator** 视角来禁用一个或多个快速启动。

先决条件

- 有集群管理员权限并登录到 web 控制台。

流程

1. 在 **Administrator** 视角中，进入 **Administration → Cluster Settings**。
2. 在 **Cluster Settings** 页面中，点 **Configuration** 选项卡。
3. 在 **Configuration** 页面中，点带有描述 **operator.openshift.io** 的 **Console** 配置资源。



4. 从 **Action** 下拉列表中，选择 **Customize**，以打开 **Cluster 配置**页面。

5. 在 **General** 选项卡中，在 **Quick start** 部分中，您可以在 **Enabled** 或 **Disabled** 列表中选择项目，并使用箭头按钮将它们从一个列表中移到另一个列表中。

   - 要启用或禁用单个快速启动，请点快速启动，然后使用单个箭头按钮将快速启动移到适当的列表中。
   - 要一次启用或禁用多个快速启动，请按 Ctrl 并点击您要移动的快速启动。然后，使用单箭头按钮将快速启动移到适当的列表中。
   - 要一次启用或禁用所有快速启动，请点双箭头按钮将所有快速开始移到适当的列表中。

第 6 章 在 OPENSHIFT CONTAINER PLATFORM 中定制 WEB 控制台

您可以对 OpenShift Container Platform web 控制台进行定制，如设置自定义徽标、产品名、链接、通知标语和命令行下载。这在您需要定制 Web 控制台以满足具体公司或政府要求时特别有用。

6.1. 添加自定义徽标和产品名称

您可以通过添加自定义徽标或自定义产品名称来创建自定义品牌。因为这些设置相互独立，因此可以两者都设置或只设置其中的一个。

先决条件

- 您必须具有管理员特权。
- 创建一个要使用的徽标文件。徽标可以是通用图像格式的文件，包括 GIF 、JPG 、PNG 或 SVG ，并有 **max-height** 为 **60px**的限制。由于 **ConfigMap** 对象大小的约束，图像大小不能超过 1 MB。

流程

1. 在 **openshift-config** 命名空间中将您的徽标文件导入到配置映射中：

```
$ oc create configmap console-custom-logo --from-file /path/to/console-custom-logo.png -n
openshift-config
```

提示

您还可以应用以下 YAML 来创建配置映射：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: console-custom-logo
  namespace: openshift-config
binaryData:
  console-custom-logo.png: <base64-encoded_logo> ... 1
```

1 提供有效的 base64 编码徽标。

2. 编辑 web 控制台的 Operator 配置使其包含 **customLogoFile** 和 **customProductName**：

```
$ oc edit consoles.operator.openshift.io cluster
```

```
apiVersion: operator.openshift.io/v1
kind: Console
metadata:
  name: cluster
spec:
  customization:
    customLogoFile:
```

```
    key: console-custom-logo.png
    name: console-custom-logo
  customProductName: My Console
```

更新 Operator 配置后，它将会把自定义的 logo 配置映射同步到控制台命名空间中，并将其挂载到 console pod 并重新部署。

3. 检查操作是否成功。如果有任何问题，控制台集群 Operator 将报告 **Degraded** 状态，控制台 Operator 配置也会报告 **CustomLogoDegraded** 状态，但状态类似于 **KeyOrFilenameInvalid** 或 **NoImageProvided**。
运行以下命令检查 **clusteroperator**：

```
$ oc get clusteroperator console -o yaml
```

运行以下命令检查 console Operator 配置：

```
$ oc get consoles.operator.openshift.io -o yaml
```

6.2. 在 WEB 控制台中创建自定义链接

先决条件

- 您必须具有管理员特权。

流程

1. 在 **Administration → Custom Resource Definitions** 中点 **ConsoleLink**。
2. 选择 **Instances** 标签
3. 点击 **Create Console Link** 并编辑文件：

```
apiVersion: console.openshift.io/v1
kind: ConsoleLink
metadata:
  name: example
spec:
  href: 'https://www.example.com'
  location: HelpMenu 1
  text: Link 1
```

1 有效的位置设置为 **HelpMenu**、**UserMenu**、**ApplicationMenu** 和 **NamespaceDashboard**。

要使自定义链接出现在所有命名空间中，请按照以下示例操作：

```
apiVersion: console.openshift.io/v1
kind: ConsoleLink
metadata:
  name: namespaced-dashboard-link-for-all-namespaces
spec:
```

```
  href: 'https://www.example.com'
  location: NamespaceDashboard
  text: This appears in all namespaces
```

要使自定义链接只出现在某些命名空间中，请按照以下示例操作：

```
apiVersion: console.openshift.io/v1
kind: ConsoleLink
metadata:
  name: namespaced-dashboard-for-some-namespaces
spec:
  href: 'https://www.example.com'
  location: NamespaceDashboard
  # This text will appear in a box called "Launcher" under "namespace" or "project" in the web
console
  text: Custom Link Text
  namespaceDashboard:
    namespaces:
    # for these specific namespaces
    - my-namespace
    - your-namespace
    - other-namespace
```

要使自定义链接出现在应用程序菜单中，请按照以下示例操作：

```
apiVersion: console.openshift.io/v1
kind: ConsoleLink
metadata:
  name: application-menu-link-1
spec:
  href: 'https://www.example.com'
  location: ApplicationMenu
  text: Link 1
  applicationMenu:
    section: My New Section
    # image that is 24x24 in size
    imageURL: https://via.placeholder.com/24
```

4. 点击 **Save** 以应用您的更改。

6.3. 自定义控制台路由

对于 **console** 和 **downloads** 路由，自定义路由功能使用 **ingress** 配置路由 API。如果 **console** 自定义路由在 **ingress** 配置和 **console-operator** 配置中都设置时，新的 **ingress** 配置自定义路由配置有高的优先级。带有 **console-operator** 配置的路由配置已弃用。

6.3.1. 自定义控制台路由

您可以通过在集群 **Ingress** 配置的 **spec.componentRoutes** 字段中设置自定义主机名和 TLS 证书来自定义控制台路由。

先决条件

- 已使用具有管理特权的用户身份登录集群。

- 您已在 **openshift-config** 命名空间中创建了包含 TLS 证书和密钥的 secret。如果自定义主机名后缀的域与集群域后缀不匹配，则需要此项。如果后缀匹配，secret 是可选的。

  提示

  您可以使用 **oc create secret tls** 命令创建 TLS secret。

流程

1. 编辑集群 **Ingress** 配置：

```
$ oc edit ingress.config.openshift.io cluster
```

2. 设置自定义主机名以及可选的服务证书和密钥：

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  componentRoutes:
    - name: console
      namespace: openshift-console
      hostname: <custom_hostname> 1
      servingCertKeyPairSecret:
        name: <secret_name> 2
```

1. 自定义主机名.
2. 对 **openshift-config** 命名空间中的 secret 的引用，该 secret 包含 TLS 证书 (**tls.crt**) 和密钥 (**tls.key**)。如果自定义主机名后缀的域与集群域后缀不匹配，则需要此项。如果后缀匹配，secret 是可选的。

3. 保存文件以使改变生效。

6.3.2. 自定义下载路由

您可以通过在集群 **Ingress** 配置的 **spec.componentRoutes** 字段中设置自定义主机名和 TLS 证书来自定义下载路由。

先决条件

- 已使用具有管理特权的用户身份登录集群。
- 您已在 **openshift-config** 命名空间中创建了包含 TLS 证书和密钥的 secret。如果自定义主机名后缀的域与集群域后缀不匹配，则需要此项。如果后缀匹配，secret 是可选的。

  提示

  您可以使用 **oc create secret tls** 命令创建 TLS secret。

流程

1. 编辑集群 **Ingress** 配置：

```
$ oc edit ingress.config.openshift.io cluster
```

2. 设置自定义主机名以及可选的服务证书和密钥：

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  componentRoutes:
    - name: downloads
      namespace: openshift-console
      hostname: <custom_hostname> 1
      servingCertKeyPairSecret:
        name: <secret_name> 2
```

1. 自定义主机名.
2. 对 **openshift-config** 命名空间中的 secret 的引用，该 secret 包含 TLS 证书 (**tls.crt**) 和密钥 (**tls.key**)。如果自定义主机名后缀的域与集群域后缀不匹配，则需要此项。如果后缀匹配，secret 是可选的。

3. 保存文件以使改变生效。

6.4. 自定义登录页面

使用自定义登录页面创建服务条款信息。如果您使用第三方登录提供程序（如 GitHub 或 Google），在将用户信任并期望它重定向到认证提供程序之前，自定义登录页面也会很有用。您还可以在验证过程中显示自定义的错误页。

注意

自定义错误模板仅限于使用重定向的身份提供程序（IDP），如请求标头和基于 OIDC 的操作。它对使用直接密码身份验证的 IDP （如 LDAP 和 htpasswd）没有影响。

先决条件

- 您必须具有管理员特权。

流程

1. 运行以下命令来创建您可以修改的模板：

```
$ oc adm create-login-template > login.html
```

```
$ oc adm create-provider-selection-template > providers.html
```

```
$ oc adm create-error-template > errors.html
```

2. 创建 secret:

```
$ oc create secret generic login-template --from-file=login.html -n openshift-config
```

```
$ oc create secret generic providers-template --from-file=providers.html -n openshift-config
```

```
$ oc create secret generic error-template --from-file=errors.html -n openshift-config
```

3. 运行：

```
$ oc edit oauths cluster
```

4. 更新规格：

```
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
# ...
spec:
  templates:
    error:
      name: error-template
    login:
      name: login-template
    providerSelection:
      name: providers-template
```

运行 **oc explain oauths.spec.templates** 以了解选项。

6.5. 为外部日志链接定义模板

如果您连接到可帮助您浏览日志的服务，但需要以特定的方式生成 URL，则可以为链接定义一个模板。

先决条件

- 您必须具有管理员特权。

流程

1. 在 **Administration → Custom Resource Definitions**中点 **ConsoleExternalLogLink**。
2. 选择 **Instances** 标签
3. 点击 **Create Console External Log Link**并编辑文件：

```
apiVersion: console.openshift.io/v1
kind: ConsoleExternalLogLink
metadata:
  name: example
spec:
  hrefTemplate: >-
    https://example.com/logs?
```

```
resourceName=${resourceName}&containerName=${containerName}&resourceNamespace=${resourceNamespace}&podLabels=${podLabels}
  text: Example Logs
```

6.6. 创建自定义通知标语

先决条件

- 您必须具有管理员特权。

流程

1. 在 **Administration → Custom Resource Definitions** 中点 **ConsoleNotification**。
2. 选择 **Instances** 标签
3. 点击 **Create Console Notification** 并编辑文件：

```
apiVersion: console.openshift.io/v1
kind: ConsoleNotification
metadata:
  name: example
spec:
  text: This is an example notification message with an optional link.
  location: BannerTop (1)
  link:
    href: 'https://www.example.com'
    text: Optional link text
  color: '#fff'
  backgroundColor: '#0088ce'
```

(1) 有效的位置设置为 **BannerTop**、**BannerBottom** 和 **BannerTopBottom**。

4. 点 **Create** 以应用您的更改。

6.7. 自定义 CLI 下载

您可以使用自定义链接文本和 URL 来配置用于下载 CLI 的链接。它们可以直接指向软件包的文件或提供软件包的外部页面。

先决条件

- 您必须具有管理员特权。

流程

1. 进入 **Administration → Custom Resource Definitions**。
2. 从 Custom Resource Definitions (CRDs) 列表中选 **ConsoleCLIDownload** 。
3. 点 **YAML** 标签页，然后进行编辑：

```
apiVersion: console.openshift.io/v1
kind: ConsoleCLIDownload
metadata:
  name: example-cli-download-links
spec:
  description: |
    This is an example of download links
  displayName: example
  links:
  - href: 'https://www.example.com/public/example.tar'
    text: example for linux
  - href: 'https://www.example.com/public/example.mac.zip'
    text: example for mac
  - href: 'https://www.example.com/public/example.win.zip'
    text: example for windows
```

4. 点 **Save** 按钮。

6.8. 在 KUBERNETES 资源中添加 YAML 示例

您可以随时动态地将 YAML 示例添加到任何 Kubernetes 资源中。

先决条件

- 您必须具有集群管理员特权。

流程

1. 在 **Administration → Custom Resource Definitions** 中点 **ConsoleYAMLSample**。
2. 点 **YAML** 并编辑该文件：

```
apiVersion: console.openshift.io/v1
kind: ConsoleYAMLSample
metadata:
  name: example
spec:
  targetResource:
    apiVersion: batch/v1
    kind: Job
  title: Example Job
  description: An example Job YAML sample
  yaml: |
    apiVersion: batch/v1
    kind: Job
    metadata:
      name: countdown
    spec:
      template:
        metadata:
          name: countdown
        spec:
          containers:
          - name: counter
```

```
      image: centos:7
      command:
      - "bin/bash"
      - "-c"
      - "for i in 9 8 7 6 5 4 3 2 1 ; do echo $i ; done"
    restartPolicy: Never
```

使用 **spec.snippet** 表示 YAML 样本不是完整的 YAML 资源定义，而是可在用户光标处的现有 YAML 文档中插入的片段。

3. 点击 **Save**。

6.9. 自定义用户透视图

OpenShift Container Platform Web 控制台默认提供两个视角，即 **Administrator（管理员视角）** 和 **Developer（开发者视角）**。根据安装的控制台插件，您可能具有更多视角。作为集群管理员，您可以为所有用户或特定用户角色显示或隐藏视角。自定义视角可确保用户只能查看适用于其角色和任务的视角。例如，您可以从非特权用户隐藏 **Administrator** 视角，以便它们无法管理集群资源、用户和项目。同样，您可以向具有 developer 角色的用户显示 **Developer** 视角，以便他们可以创建、部署和监控应用程序。

您还可以根据基于角色的访问控制 (RBAC) 自定义用户的可见性。例如，如果您自定义了需要特定权限的监控目的的视角，您可以定义视角仅对具有所需权限的用户可见。

每个视角都包括以下强制参数，您可以在 YAML 视图中编辑它们：

- **id**：定义要显示或隐藏的透视图的 ID
- **visibility** ：定义透视图的状态以及访问权限检查（如果需要）
- **State** ：定义视角是启用、禁用还是需要访问权限检查

注意

默认情况下，所有视角都已启用。当您自定义用户视角时，您的更改适用于整个集群。

6.9.1. 使用 YAML 视图自定义透视图

先决条件

- 您必须具有管理员特权。

流程

1. 在 **Administrator** 视角中，进入 **Administration → Cluster Settings**。
2. 选择 **Configuration** 选项卡，再点 **Console (operator.openshift.io)** 资源。
3. 点 **YAML** 选项卡并进行自定义：
   a. 要启用或禁用一个视角，请插入**添加用户视角**的代码片段，并根据需要编辑 YAML 代码：

```
apiVersion: operator.openshift.io/v1
kind: Console
metadata:
  name: cluster
```

```
spec:
  customization:
    perspectives:
      - id: admin
        visibility:
          state: Enabled
      - id: dev
        visibility:
          state: Enabled
```

b. 要根据 RBAC 权限隐藏视图，请插入**隐藏用户视角**的代码片段，并根据需要编辑 YAML 代码：

```
apiVersion: operator.openshift.io/v1
kind: Console
metadata:
  name: cluster
spec:
  customization:
    perspectives:
      - id: admin
        requiresAccessReview:
          - group: rbac.authorization.k8s.io
            resource: clusterroles
            verb: list
      - id: dev
        state: Enabled
```

c. 要根据您的需要自定义视角，请创建自己的 YAML 片断：

```
apiVersion: operator.openshift.io/v1
kind: Console
metadata:
  name: cluster
spec:
  customization:
    perspectives:
      - id: admin
        visibility:
          state: AccessReview
          accessReview:
            missing:
              - resource: deployment
                verb: list
            required:
              - resource: namespaces
                verb: list
      - id: dev
        visibility:
          state: Enabled
```

4. 点击 **Save**。

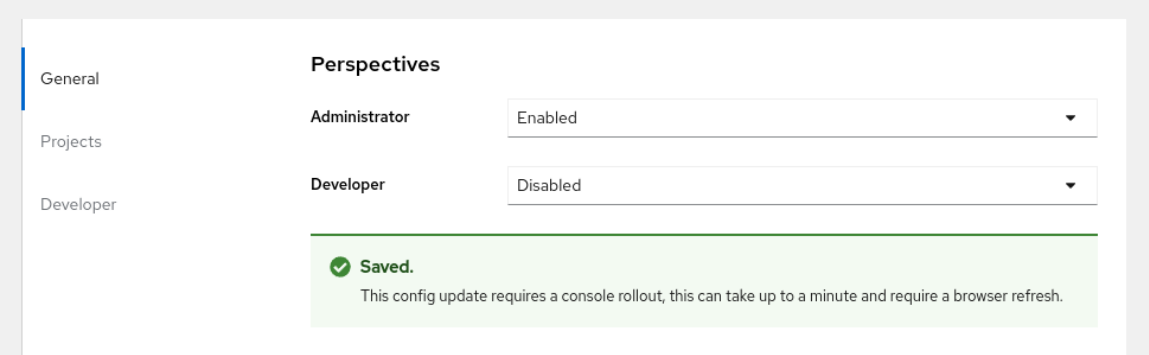
6.9.2. 使用表单视图自定义透视图

先决条件

- 您必须具有管理员特权。

流程

1. 在 **Administrator** 视角中，进入 **Administration → Cluster Settings**。
2. 选择 **Configuration** 选项卡，再点 **Console (operator.openshift.io)** 资源。
3. 在页面右侧点击 **Actions → Customize**。
4. 在**常规设置**中，通过从下拉列表中选择以下选项之一来自定义视角：
   - **启用** ：为所有用户启用视角
   - **仅对特权用户可见** ：启用可以列出所有命名空间的用户的视角
   - **仅对非特权用户可见** ：启用对无法列出所有命名空间的用户的视角
   - **禁用**：禁用所有用户的视角
     这时将打开通知，以确认您的更改已保存。



注意

当您自定义用户视角时，您的更改会自动保存并在浏览器刷新后生效。

6.10. 开发人员目录和子目录自定义

作为集群管理员，您可以组织和管理 Developer 目录或其子目录。您可以启用或禁用子目录类型或禁用整个开发人员目录。

developerCatalog.types 对象包括以下参数，您必须在 YAML 视图中使用它们：

- **state**：定义开发人员目录类型的列表是否应启用或禁用。
- **enabled**：定义用户可见的开发人员目录类型 (sub-catalogs) 列表。
- **disabled**：定义用户不可见的开发人员目录类型 (sub-catalogs) 列表。

您可以使用 YAML 视图或表单视图启用或禁用以下开发人员目录类型 (sub-catalogs)。

- **构建器镜像**
- **模板**

- **Devfiles**
- **Samples**
- **Helm Charts**
- **事件源**
- **事件 Sinks**
- **Operator Backed**

6.10.1. 使用 YAML 视图自定义开发人员目录或其子目录

您可以通过编辑 YAML 视图中的 YAML 内容来自定义开发人员目录。

先决条件

- 具有集群管理员权限的 OpenShift Web 控制台会话。

流程

1. 在 Web 控制台的 **Administrator 视角中**，导航到 **Administration → Cluster Settings**。
2. 选择 **Configuration** 选项卡，点 **Console (operator.openshift.io)** 资源并查看 **Details** 页面。
3. 点 **YAML** 选项卡打开编辑器，并根据需要编辑 YAML 内容。
   例如，要禁用开发人员目录类型，请插入以下代码片段，以定义已禁用开发人员目录资源的列表：

```
apiVersion: operator.openshift.io/v1
kind: Console
metadata:
  name: cluster
...
spec:
  customization:
    developerCatalog:
      categories:
      types:
        state: Disabled
        disabled:
          - BuilderImage
          - Devfile
          - HelmChart
...
```

4. 点击 **Save**。

注意

默认情况下，开发人员目录类型在 Web 控制台的 Administrator 视图中启用。

6.10.2. 使用表单视图自定义开发人员目录或其子目录

您可以使用 Web 控制台中的表单视图自定义开发人员目录。

先决条件

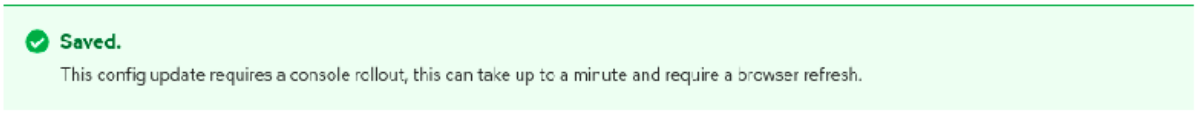
- 具有集群管理员权限的 OpenShift Web 控制台会话。
- Developer 视角被启用。

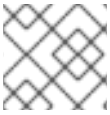
流程

1. 在 **Administrator** 视角中，进入 **Administration → Cluster Settings**。
2. 选择 **Configuration** 选项卡，再点 **Console (operator.openshift.io)** 资源。
3. 点 **Actions → Customize**。
4. 在 **Pre-pinned 导航项**、**Add page** 和 **Developer Catalog** 部分中启用或禁用项目。

验证

自定义开发人员目录后，您的更改会自动保存到系统中，并在刷新后在浏览器中生效。





注意

作为管理员，您可以定义所有用户默认出现的导航项。您还可以重新排序导航项。

提示

您可以使用类似的流程来自定义 Web UI 项目，如快速启动、集群角色和操作。

6.10.2.1. YAML 文件更改示例

您可以在 YAML 编辑器中动态添加以下代码片段，以自定义开发人员目录。

通过将 *state* 类型设置为 **Enabled**，使用以下代码片段显示所有子目录：

```
apiVersion: operator.openshift.io/v1
kind: Console
metadata:
  name: cluster
...
spec:
  customization:
    developerCatalog:
      categories:
      types:
        state: Enabled
```

使用以下代码片段，通过将 *state* 类型设置为 **Disabled** 来禁用所有子目录：

```
apiVersion: operator.openshift.io/v1
kind: Console
metadata:
  name: cluster
...
spec:
  customization:
    developerCatalog:
      categories:
      types:
        state: Disabled
```

当集群管理员定义了子目录列表（在 Web 控制台中启用）时，使用以下代码片段。

```
apiVersion: operator.openshift.io/v1
kind: Console
metadata:
  name: cluster
...
spec:
  customization:
    developerCatalog:
      categories:
      types:
        state: Enabled
        enabled:
          - BuilderImage
          - Devfile
          - HelmChart
          - ...
```

第 7 章 动态插件

7.1. 动态插件概述

7.1.1. 关于动态插件

动态插件作为工作负载在集群中部署。它们允许您在运行时为控制台用户界面添加自定义页面和其他扩展。**ConsolePlugin** 自定义资源使用控制台注册插件，集群管理员在 **console-operator** 配置中启用了插件。

7.1.2. 主要特性

通过动态插件，您可以对 OpenShift Container Platform 体验进行以下自定义：

- 添加自定义页面。
- 增加了除管理员和开发人员之外的其他视角。
- 添加导航项。
- 在资源页面中添加制表符和操作。

7.1.3. 常规指南

在创建插件时，请遵循以下常规准则：

- 构建和运行插件需要 **Node.js** 和 **yarn**。
- 为您的 CSS 类名称加上插件名称前缀，以避免冲突。例如，**my-plugin__heading** 和 **my-plugin__icon**。
- 与其他控制台页面保持一致的外观、感觉和行为。
- 在创建插件时遵循 react-i18next 指南。您可以使用类似以下示例中的 **useTranslation** hook:

```
conster Header: React.FC = () => {
  const { t } = useTranslation('plugin__console-demo-plugin');
  return <h1>{t('Hello, World!')}</h1>;
};
```

- 避免可能影响插件组件之外的标记的选择器，如元素选择器。这些不是 API，可能随时更改。使用它们可能会破坏插件。避免选择器，如元素选择器，它们可能会影响插件组件之外的标记。
- 为插件 Web 服务器提供的所有资产，使用 **Content-Type** 响应标头提供有效的 JavaScript 多用途互联网邮件扩展(MIME)类型。每个插件部署都应该包含一个 Web 服务器，用于托管给定插件生成的资产。

PatternFly 指南

在创建插件时，请按照以下使用 PatternFly 的准则进行以下操作：

- 使用 PatternFly 组件和 PatternFly CSS 变量。SDK 提供了核心 PatternFly 组件。使用 PatternFly 组件和变量可帮助您的插件在将来的控制台版本中保持一致。
  - 如果您使用 OpenShift Container Platform 版本 4.14 及更早版本，请使用 Patternfly 4.x。

- 如果使用 OpenShift Container Platform 4.15 或更高版本，请使用 Patternfly 5.x。

- 您可以按照 PatternFly 的可访问性基础，使您的插件能被访问。
- 避免使用其他 CSS 库，如 Bootstrap 或 Tailwind。它们可能会与 PatternFly 冲突，不会与控制台的外观和感觉相匹配。插件应仅包含特定于其用户界面的样式，以便在基本 PatternFly 样式上评估。避免在插件中导入 **@patternfly/react-styles/*/.css** 或来自 **@patternfly/patternfly** 软件包的任何样式。
- 控制台应用程序负责为所有支持的 PatternFly 版本载入基本风格。

7.2. 动态插件入门

要开始使用动态插件，您必须设置您的环境来编写新的 OpenShift Container Platform 动态插件。有关如何编写新插件的示例，请参阅 Adding a tab to the pods 页 。

7.2.1. 动态插件开发

您可以使用本地开发环境运行插件。OpenShift Container Platform Web 控制台在一个连接到您登录的集群的容器中运行。

先决条件

- 您必须有一个 OpenShift 集群正在运行。
- 已安装 OpenShift CLI (**oc**)。
- 需要安装 **yarn**。
- 已安装并运行 Docker v3.2.0 或更新版本或 Podman。

流程

1. 在终端中，运行以下命令使用 yarn 安装插件的依赖项。

```
$ yarn install
```

2. 安装后，运行以下命令来启动 yarn。

```
$ yarn run start
```

3. 在另一个终端窗口中，通过 CLI 登录 OpenShift Container Platform。

```
$ oc login
```

4. 运行以下命令，在连接到您登录的集群的容器中运行 OpenShift Container Platform Web 控制台：

```
$ yarn run start-console
```

验证

- 访问 localhost:9000 以查看正在运行的插件。检查 **window.SERVER_FLAGS.consolePlugins** 的值，以查看在运行时加载的插件列表。

7.3. 在集群中部署插件

您可以将插件部署到 OpenShift Container Platform 集群。

7.3.1. 使用 Docker 构建镜像

要在集群中部署插件，您需要构建镜像并将其推送到镜像 registry。

流程

1. 使用以下命令构建镜像：

```
$ docker build -t quay.io/my-repositroy/my-plugin:latest .
```

2. 可选：如果要测试您的镜像，请运行以下命令：

```
$ docker run -it --rm -d -p 9001:80 quay.io/my-repository/my-plugin:latest
```

3. 运行以下命令推送镜像：

```
$ docker push quay.io/my-repository/my-plugin:latest
```

7.3.2. 在集群中部署插件

在将镜像推送到 registry 后，您可以将插件部署到集群中。

流程

1. 要将插件部署到集群中，请使用插件名称作为 Helm 发行版本名称安装 Helm chart，作为 Helm 发行版本名称，或由 **-n** 命令行选项指定的现有命名空间。使用以下命令，提供 **plugin.image** 参数中镜像的位置：

```
$ helm upgrade -i  my-plugin charts/openshift-console-plugin -n my-plugin-namespace --
create-namespace --set plugin.image=my-plugin-image-location
```

其中：

n <my-plugin-namespace>
: 指定要将插件部署到的现有命名空间。

--create-namespace
: 可选：如果部署到新命名空间，请使用此参数。

--set plugin.image=my-plugin-image-location
: 指定 **plugin.image** 参数中镜像的位置。

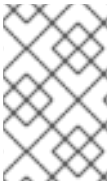
2. 可选： 您可以使用 **charts/openshift-console-plugin/values.yaml** 文件中的一组支持的参数来指定任何其他参数。

```
plugin:
```

```
  name: ""
  description: ""
  image: ""
  imagePullPolicy: IfNotPresent
  replicas: 2
  port: 9443
  securityContext:
    enabled: true
  podSecurityContext:
    enabled: true
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containerSecurityContext:
    enabled: true
    allowPrivilegeEscalation: false
    capabilities:
      drop:
        - ALL
  resources:
    requests:
      cpu: 10m
      memory: 50Mi
  basePath: /
  certificateSecretName: ""
  serviceAccount:
    create: true
    annotations: {}
    name: ""
  patcherServiceAccount:
    create: true
    annotations: {}
    name: ""
  jobs:
    patchConsoles:
      enabled: true
      image: "registry.redhat.io/openshift4/ose-tools-
rhel8@sha256:e44074f21e0cca6464e50cb6ff934747e0bd11162ea01d522433a1a1ae116103"

      podSecurityContext:
        enabled: true
        runAsNonRoot: true
        seccompProfile:
          type: RuntimeDefault
      containerSecurityContext:
        enabled: true
        allowPrivilegeEscalation: false
        capabilities:
          drop:
            - ALL
      resources:
        requests:
          cpu: 10m
          memory: 50Mi
```

验证

- 从 **Administration → Cluster Settings → Configuration → Console operator.openshift.io → Console plugins** 或访问 **Overview** 页面来查看启用的插件列表。



注意

显示新插件配置可能需要几分钟时间。如果没有看到插件，则在最近启用了插件时，可能需要刷新浏览器。如果您在运行时收到任何错误，请在浏览器开发人员工具中检查 JS 控制台，以查看插件代码中的任何错误。

7.3.3. 插件服务代理

如果您需要从插件向集群服务发出 HTTP 请求，您可以使用 **spec.proxy** 数组字段在 **ConsolePlugin** 资源中声明服务代理。控制台后端公开 **/api/proxy/plugin/<plugin-name>/<proxy-alias>/<request-path>?<optional-query-parameters>** 端点来代理插件和服务之间的通信。代理请求默认使用 *服务 CA 捆绑包*。服务必须使用 HTTPS。

注意

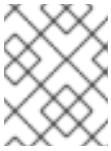
该插件必须使用 **consolefetch** API 从 JavaScript 代码发出请求，或者一些请求可能会失败。如需更多信息，请参阅 "Dynamic plugin API"。

对于每个条目，您必须分别在 **endpoint** 和 **alias** 字段中指定代理的端点和别名。对于 Service 代理类型，您必须将端点 **type** 字段设置为 **Service**，**service** 必须包含 **name**, **namespace**, 和 **port** 字段的值。例如，**/api/proxy/plugin/helm/helm-charts/releases?limit=10** 是代理请求路径，它来自带有一个 **helm-charts** 服务的 **helm** 插件，列出 10 个 helm release。

服务代理示例

```
apiVersion: console.openshift.io/v1
kind: ConsolePlugin
metadata:
  name:<plugin-name>
spec:
  proxy:
  - alias: helm-charts (1)
    authorization: UserToken (2)
    caCertificate: '-----BEGIN CERTIFICATE-----\nMIID....'en (3)
    endpoint: (4)
      service:
        name: <service-name>
        namespace: <service-namespace>
        port: <service-port>
      type: Service
```

1. 代理的别名。
2. 如果服务代理请求必须包含登录用户的 OpenShift Container Platform 访问令牌，您必须将 authorization 字段设置为 **UserToken**。



注意

如果服务代理请求不包含登录用户的 OpenShift Container Platform 访问令牌，请将 authorization 字段设置为 **None**。

3 如果服务使用自定义服务 CA，**caCertificate** 字段必须包含证书捆绑包。

4 代理的端点。

其他资源

- 服务 CA 证书
- 使用服务提供的证书 secret 保护服务流量
- 动态插件 API

7.3.4. 在浏览器中禁用您的插件

控制台用户可以使用 **disable-plugins** 查询参数来禁用通常会在运行时加载的特定或所有动态插件。

流程

- 要禁用特定的插件，请从以逗号分隔的插件名称列表中删除您要禁用的插件。
- 要禁用所有插件，请在 **disable-plugins** 查询参数中有一个空字符串。

注意

集群管理员可以在 web 控制台的 **Cluster Settings** 页面中禁用插件

7.3.5. 其他资源

- 了解 Helm

7.4. 动态插件示例

在操作示例前，请按照动态插件开发中的步骤来验证插件是否正常工作

7.4.1. 在 Pod 页面中添加标签页

您可以对 OpenShift Container Platform Web 控制台进行不同的自定义配置。以下流程在 **Pod Details** 页中添加一个标签页，作为插件的一个示例扩展。

注意

OpenShift Container Platform Web 控制台在一个连接到您登录的集群的容器中运行。有关在创建自己的前测试插件的信息，请参阅"动态插件开发"。

流程

1. 访问 **console-plugin-template** 存储库，其中包含用于在新标签页中创建插件的模板。

重要

红帽不支持自定义插件代码。对于插件，只有合作社区支持。

2. 点 **Use this template** → ***Create new repository***，为模板创建一个 GitHub 存储库。
3. 使用插件的名称替换新存储库。
4. 将新存储库克隆到本地机器，以便您可以编辑代码。
5. 编辑 **package.json** 文件，将插件的元数据添加到 **consolePlugin** 声明中。例如：

```
"consolePlugin": {
  "name": "my-plugin", (1)
  "version": "0.0.1", (2)
  "displayName": "My Plugin", (3)
  "description": "Enjoy this shiny, new console plugin!", (4)
  "exposedModules": {
    "ExamplePage": "./components/ExamplePage"
  },
  "dependencies": {
    "@console/pluginAPI": "/*"
  }
}
```

(1) 更新插件的名称。

(2) 更新版本。

(3) 更新插件的显示名称。

(4) 使用有关插件的同步更新描述。

6. 在 **console-extensions.json** 文件中添加以下内容：

```
{
  "type": "console.tab/horizontalNav",
  "properties": {
    "page": {
      "name": "Example Tab",
      "href": "example"
    },
    "model": {
      "group": "core",
      "version": "v1",
      "kind": "Pod"
    },
    "component": { "$codeRef": "ExampleTab" }
  }
}
```

7. 编辑 **package.json** 文件以包括以下更改：

```
      "exposedModules": {
```

```
        "ExamplePage": "./components/ExamplePage",
        "ExampleTab": "./components/ExampleTab"
    }
```

8. 通过创建新文件 **src/components/ExampleTab.tsx** 并添加以下脚本，在 Pod 页面上的一个新自定义标签页中写入信息：

```
import * as React from 'react';

export default function ExampleTab() {
  return (
    <p>This is a custom tab added to a resource using a dynamic plugin.</p>
  );
}
```

9. 使用插件名称作为 Helm 发行版本名称安装 Helm Chart，或由 **-n** 命令行选项指定的现有命名空间，以便在集群中部署插件。使用以下命令，提供 **plugin.image** 参数中镜像的位置：

```
$ helm upgrade -i  my-plugin charts/openshift-console-plugin -n my-plugin-namespace --create-namespace --set plugin.image=my-plugin-image-location
```

注意

有关在集群中部署插件的更多信息，请参阅"在集群中部署插件"。

验证

- 访问 Pod 页面查看添加的选项卡。

7.5. 动态插件参考

您可以添加允许您自定义插件的扩展。这些扩展随后会在运行时加载到控制台。

7.5.1. 动态插件扩展类型

console.action/filter
ActionFilter 可用于过滤操作。

Name	值类型	选填	描述
contextId	**string**	否	上下文 ID 有助于将贡献操作的范围缩小到应用的特定区域。示例包括 **topology** 和 **helm**。

Name	值类型	选填	描述
filter	**CodeRef<(scope: any, action: Action) ⇒ boolean>**	否	将根据某些条件过滤操作的功能。 **scope**：为其提供操作的范围。如果要使用 Pod 横向自动扩展 (HPA) 从部署中删除 **ModifyCount** 操作，则可能需要 hook。

console.action/group
ActionGroup 贡献一个操作组，也可以是一个子菜单

Name	值类型	选填	描述
id	**string**	否	用于识别操作部分的 ID。
label	**string**	是	UI 中显示的标签。子菜单是必需的。
submenu	**布尔值**	是	此组是否应显示为子菜单。
insertBefore	**string | string[]**	是	在此处引用的项目前插入此项。对于数组，使用按顺序找到的第一个。
insertAfter	**string | string[]**	是	在此处引用的项目后插入此项。对于数组，使用按顺序找到的第一个。**insertBefore** 值具有优先权。

console.action/provider
ActionProvider 贡献了一个 hook，用于返回特定上下文的操作列表。

Name	值类型	选填	描述
contextId	**string**	否	上下文 ID 有助于将贡献操作的范围缩小到应用的特定区域。示例包括 **topology** 和 **helm**。

Name	值类型	选填	描述
provider	**CodeRef<ExtensionHook<Action[], any>>**	否	一个 React hook，用于返回给定范围的操作。如果 **contextId** = **resource**，则范围始终为 Kubernetes 资源对象。

console.action/resource-provider
ResourceActionProvider 贡献 hook，用于返回特定资源模型的操作列表。

Name	值类型	选填	描述
model	**ExtensionK8sKindVersionModel**	否	此提供程序提供操作的模型。
provider	**CodeRef<ExtensionHook<Action[], any>>**	否	反应 hook，它返回给定资源模型的操作

console.alert-action
当控制台根据其 **rule.name** 值观察特定 Prometheus 警报时，此扩展可用于触发特定操作。

Name	值类型	选填	描述
alert	**string**	否	由 **alert.rule.name** 属性定义的警报名称
text	**string**	否	
action	**CodeRef<(alert: any) ⇒ void>**	否	执行副作用的功能

console.catalog/item-filter
此扩展可用于插件，以生成一个可以过滤特定目录项的处理程序。例如，插件可以贡献从特定提供程序过滤 helm chart 的过滤器。

Name	值类型	选填	描述
catalogId	**string | string[]**	否	此提供程序参与的目录的唯一标识符。
type	**string**	否	目录项目类型的类型 ID。

Name	值类型	选填	描述
filter	**CodeRef<(item: CatalogItem) ⇒ boolean>**	否	过滤特定类型的项目。值是采用 **CatalogItem[]** 且根据过滤器标准返回子集的功能。

console.catalog/item-metadata
此扩展可用于贡献向特定目录项添加额外的元数据的供应商。

Name	值类型	选填	描述
catalogId	**string | string[]**	否	此提供程序参与的目录的唯一标识符。
type	**string**	否	目录项目类型的类型 ID。
provider	**CodeRef<ExtensionHook<CatalogItemMetadataProviderFunction, CatalogExtensionHookOptions>>**	否	返回一个将用来向特定类型的目录项提供元数据的 hook。

console.catalog/item-provider
此扩展允许插件为目录项类型贡献供应商。例如，Helm 插件可以添加一个供应商来获取所有 Helm Charts。此扩展也可以被其他插件使用，来向特定的目录项类型添加更多项目。

Name	值类型	选填	描述
catalogId	**string | string[]**	否	此提供程序参与的目录的唯一标识符。
type	**string**	否	目录项目类型的类型 ID。
title	**string**	否	目录项提供程序的标题
provider	**CodeRef<ExtensionHook<CatalogItem<any>[], CatalogExtensionHookOptions>>**	否	为目录获取项目并进行规范化。值是对 hook 做出反应。

Name	值类型	选填	描述
priority	**number**	是	此提供程序的优先级。默认值为 **0**。优先级更高的提供程序可能会覆盖由其他供应商提供的目录项。

console.catalog/item-type
此扩展允许插件生成新的目录项。例如，Helm 插件可将新的目录项类型定义为它希望向 Developer Catalog 贡献的 HelmCharts。

Name	值类型	选填	描述
type	**string**	否	目录项的类型。
title	**string**	否	目录项的标题。
catalogDescription	**string | CodeRef<React.ReactNode>**	是	特定与目录类型的描述。
typeDescription	**string**	是	目录项类型的描述。
filters	**CatalogItemAttribute []**	是	特定于目录项的自定义过滤器。
groupings	**CatalogItemAttribute []**	是	特定于目录项的自定义分组。

console.catalog/item-type-metadata
此扩展允许插件为任何目录项类型贡献额外的元数据，如自定义过滤器或分组。例如，插件可以为根据 Chart 供应商过滤的 HelmCharts 附加自定义过滤器。

Name	值类型	选填	描述
type	**string**	否	目录项的类型。
filters	**CatalogItemAttribute []**	是	特定于目录项的自定义过滤器。
groupings	**CatalogItemAttribute []**	是	特定于目录项的自定义分组。

console.cluster-overview/inventory-item
将新清单项添加到集群概览页面中。

Name	值类型	选填	描述
component	**CodeRef<React.ComponentType<{}>>**	否	要呈现的组件。

console.cluster-overview/multiline-utilization-item
添加新的集群概述多行使用项。

Name	值类型	选填	描述
title	**string**	否	使用项目的标题。
getUtilizationQueries	**CodeRef<GetMultilineQueries>**	否	Prometheus 使用率查询。
humanize	**CodeRef<Humanize>**	否	将 Prometheus 数据转换为人类可读形式。
TopConsumerPopovers	**CodeRef<React.ComponentType<TopConsumerPopoverProps>[]>**	是	显示顶部消费者弹出而不是普通值

console.cluster-overview/utilization-item
添加新的集群概览使用率项目。

Name	值类型	选填	描述
title	**string**	否	使用项目的标题。
getUtilizationQuery	**CodeRef<GetQuery>**	否	Prometheus 使用率查询。
humanize	**CodeRef<Humanize>**	否	将 Prometheus 数据转换为人类可读形式。
getTotalQuery	**CodeRef<GetQuery>**	是	Prometheus 总计查询。
getRequestQuery	**CodeRef<GetQuery>**	是	Prometheus 请求查询。
getLimitQuery	**CodeRef<GetQuery>**	是	Prometheus 限制查询。
TopConsumerPopover	**CodeRef<React.ComponentType<TopConsumerPopoverProps>>**	是	显示顶部消费者弹出而不是普通值

console.context-provider
在 web 控制台应用程序根目录中添加新的 React 上下文提供程序。

Name	值类型	选填	描述
provider	**CodeRef<Provider<T>></**	否	上下文提供程序组件。
useValueHook	**CodeRef<() ⇒ T>**	否	上下文值的 hook。

console.dashboards/card
添加新的仪表板卡。

Name	值类型	选填	描述
tab	**string**	否	将添加到卡的仪表板标签的 ID。
position	**'LEFT' | 'RIGHT' | 'MAIN'**	否	该卡在仪表板上的网格位置。
component	**CodeRef<React.ComponentType<{}>>**	否	仪表板卡组件。
span	**OverviewCardSpan**	是	栏中卡的垂直范围。对于小屏幕会忽略，默认为 **12**。

console.dashboards/custom/overview/detail/item
在 Overview 仪表板的详情卡中添加项目。

Name	值类型	选填	描述
title	**string**	否	详情卡标题
component	**CodeRef<React.ComponentType<{}>>**	否	由 OverviewDetailItem 组件呈现的值
valueClassName	**string**	是	className 的值
isLoading	**CodeRef<() ⇒ boolean>**	是	返回组件的加载状态的功能
错误	**CodeRef<() ⇒ string>**	是	组件显示的功能返回错误

console.dashboards/overview/activity/resource

在 Overview 仪表板的活动卡中添加一个活动，其中根据监视 Kubernetes 资源触发活动。

Name	值类型	选填	描述
k8sResource	**CodeRef<FirehoseResource & { isList: true; }>**	否	要替换的 utilization 项。
component	**CodeRef<React.ComponentType<K8sActivityProps<T>>>**	否	操作组件。
isActivity	**CodeRef<(resource: T) ⇒ boolean>**	是	确定给定资源是否代表该操作的功能。如果没有定义，则每个资源都代表活动。
getTimestamp	**CodeRef<(resource: T) ⇒ Date>**	是	给定操作的时间戳，用于排序。

console.dashboards/overview/health/operator
在 Overview 仪表板的状态卡中添加一个健康子系统，其中状态源是 Kubernetes REST API。

Name	值类型	选填	描述
title	**string**	否	弹出菜单中的 Operators 部分的标题。
资源	**CodeRef<FirehoseResource[]>**	否	将获取并传递给 **healthHandler** 的 Kubernetes 资源。
getOperatorsWithStatuses	**CodeRef<GetOperatorsWithStatuses<T>>**	是	解析 Operator 的状态。
operatorRowLoader	**CodeRef<React.ComponentType<OperatorRowProps<T>>>**	是	弹出行组件的加载程序。
viewAllLink	**string**	是	链接到所有资源页面。如果没有提供，则使用资源 prop 中第一个资源的列表页面。

console.dashboards/overview/health/prometheus
在 Status 为 Prometheus 的 Overview 仪表板的状态卡中添加一个健康子系统。

Name	值类型	选填	描述
title	**string**	否	子系统的显示名称。
queries	**string[]**	否	Prometheus 查询。
healthHandler	**CodeRef<PrometheusHealthHandler>**	否	解决子系统的健康状况。
additionalResource	**CodeRef<FirehoseResource>**	是	将获取并传递给 **healthHandler** 的其他资源。
popupComponent	**CodeRef<React.ComponentType<PrometheusHealthPopupProps>>**	是	弹出式菜单内容的加载程序。如果定义，健康项表示为链接，它会打开一个带有给定内容的弹出窗口。
popupTitle	**string**	是	弹出的标题。
disallowedControlPlaneTopology	**string[]**	是	应该隐藏子系统的 control plane 拓扑。

console.dashboards/overview/health/resource
在状态仪表板的 Overview 卡中添加一个健康子系统，其中状态源是一个 Kubernetes 资源。

Name	值类型	选填	描述
title	**string**	否	子系统的显示名称。
资源	**CodeRef<WatchK8sResources<T>>**	否	将获取并传递给 **healthHandler** 的 Kubernetes 资源。
healthHandler	**CodeRef<ResourceHealthHandler<T>>**	否	解决子系统的健康状况。
popupComponent	**CodeRef<WatchK8sResults<T>>**	是	弹出式菜单内容的加载程序。如果定义，健康项表示为链接，它会打开一个带有给定内容的弹出窗口。
popupTitle	**string**	是	弹出的标题。

console.dashboards/overview/health/url
在状态仪表板的 Overview 卡中添加一个健康子系统，其中状态源是一个 Kubernetes REST API。

Name	值类型	选填	描述
title	**string**	否	子系统的显示名称。
url	**string**	否	从中获取数据的 URL。它将以基本 Kubernetes URL 作为前缀。
healthHandler	**CodeRef<URLHealthHandler<T, K8sResourceCommon | K8sResourceCommon[]>>**	否	解决子系统的健康状况。
additionalResource	**CodeRef<FirehoseResource>**	是	将获取并传递给 **healthHandler** 的其他资源。
popupComponent	**CodeRef<React.ComponentType<{ healthResult?: T; healthResultError?: any; k8sResult?: FirehoseResult<R>; }>>**	是	弹出内容的加载程序。如果定义，则一个健康项目将显示为一个链接，该链接会打开给定内容弹出窗口中。
popupTitle	**string**	是	弹出的标题。

console.dashboards/overview/inventory/item
在概述清单卡中添加资源标题。

Name	值类型	选填	描述
model	**CodeRef<T>**	否	将获取的**资源**模型。用于获取模型的 **label** 或 **abbr**。
mapper	**CodeRef<StatusGroupMapper<T, R>>**	是	将各种状态映射到组的功能。
additionalResources	**CodeRef<WatchK8sResources<R>>**	是	将获取并传递给**映射程序**函数的其他资源。

console.dashboards/overview/inventory/item/group
添加清单状态组。

Name	值类型	选填	描述
id	**string**	否	状态组的 ID。
icon	**CodeRef<React.ReactElement<any, string | React.JSXElementConstructor<any>>>**	否	响应代表状态组图标的组件。

console.dashboards/overview/inventory/item/replacement
替换概述清单卡。

Name	值类型	选填	描述
model	**CodeRef<T>**	否	将获取的**资源**模型。用于获取模型的 **label** 或 **abbr**。
mapper	**CodeRef<StatusGroupMapper<T, R>>**	是	将各种状态映射到组的功能。
additionalResources	**CodeRef<WatchK8sResources<R>>**	是	将获取并传递给**映射程序**函数的其他资源。

console.dashboards/overview/prometheus/activity/resource
在 Prometheus Overview 仪表板的 Activity 卡中添加一个活动，该仪表板根据监视 Kubernetes 资源来触发活动。

Name	值类型	选填	描述
queries	**string[]**	否	要监视的查询。
component	**CodeRef<React.ComponentType<PrometheusActivityProps>>**	否	操作组件。
isActivity	**CodeRef<(results: PrometheusResponse[]) ⇒ boolean>**	是	确定给定资源是否代表该操作的功能。如果没有定义，则每个资源都代表活动。

console.dashboards/project/overview/item
为项目概述清单卡中添加资源标题。

Name	值类型	选填	描述
model	**CodeRef<T>**	否	将获取的**资源**模型。用于获取模型的 **label** 或 **abbr**。
mapper	**CodeRef<StatusGroupMapper<T, R>>**	是	将各种状态映射到组的功能。
additionalResources	**CodeRef<WatchK8sResources<R>>**	是	将获取并传递给**映射程序**函数的其他资源。

console.dashboards/tab
添加新仪表板选项卡，它位于 Overview 选项卡的后面。

Name	值类型	选填	描述
id	**string**	否	唯一标签页标识符，用作标签链接 **href** 以及在此标签页中添加卡时。
navSection	**'home' | 'storage'**	否	选项卡所属的导航部分。
title	**string**	否	选项卡的标题。

console.file-upload
此扩展可用于为特定文件扩展提供文件丢弃操作的处理程序。

Name	值类型	选填	描述
fileExtensions	**string[]**	否	支持的文件扩展。
handler	**CodeRef<FileUploadHandler>**	否	处理文件丢弃操作的功能。

console.flag
提供对 Web 控制台功能标记的完整控制。

Name	值类型	选填	描述
handler	**CodeRef<FeatureFlagHandler>**	否	用于设置或取消设置任意功能标记。

console.flag/hookProvider
提供对带有 hook 处理程序的 Web 控制台功能标记的完整控制。

Name	值类型	选填	描述
handler	**CodeRef<FeatureFlagHandler>**	否	用于设置或取消设置任意功能标记。

console.flag/model
通过在集群中存在 **CustomResourceDefinition** (CRD) 对象来添加新的 Web 控制台功能标记。

Name	值类型	选填	描述
flag	**string**	否	在检测到 CRD 后要设置的标记名称。
model	**ExtensionK8sModel**	否	引用 CRD 的模型。

console.global-config
此扩展标识用于管理集群配置的资源。资源的链接将添加到 **Administration → Cluster Settings → Configuration** 页面中。

Name	值类型	选填	描述
id	**string**	否	集群配置资源实例的唯一标识符。
name	**string**	否	集群配置资源实例的名称。
model	**ExtensionK8sModel**	否	指代集群配置资源的模型。
namespace	**string**	否	集群配置资源实例的命名空间。

console.model-metadata
通过覆盖通过 API 发现检索并生成的值来自定义模型显示。

Name	值类型	选填	描述
model	**ExtensionK8sGroupModel**	否	要定制的模型。只能指定组或可选版本和类型。
badge	**ModelBadge**	是	是否将这个模型视为技术预览还是开发者预览。
color	**string**	是	与这个模型关联的颜色。

Name	值类型	选填	描述
label	**string**	是	覆盖标签。需要提供的 **kind**。
labelPlural	**string**	是	覆盖复数标签。需要提供的 **kind**。
abbr	**string**	是	自定义缩写。默认为 **kind** 中的所有字符的大写，最多 4 个字符。需要 **kind** 提供。

console.navigation/href
此扩展可用于贡献指向 UI 中特定链接的导航项。

Name	值类型	选填	描述
id	**string**	否	此项目的唯一标识符。
name	**string**	否	此项目的名称。
href	**string**	否	链接 **href** 值。
perspective	**string**	是	此项目所属的视角 ID。若未指定，则向默认视角。
节	**string**	是	导航此项目所属的导航部分。如果未指定，请将此项目呈现为顶级链接。
dataAttributes	**{ [key: string]: string; }**	是	在 DOM 中添加数据属性。
startsWith	**string[]**	是	当 URL 从其中一个路径之一时，将此项标记为 active。
insertBefore	**string | string[]**	是	在此处引用的项目前插入此项。对于数组，使用按顺序找到的第一个。
insertAfter	**string | string[]**	是	在此处引用的项目后插入此项。对于数组，使用按顺序找到的第一个。**insertBefore** 具有高优先级。

Name	值类型	选填	描述
namespaced	**布尔值**	是	如果为 **true**，请在末尾添加 **/ns/active-namespace**。
prefixNamespaced	**布尔值**	是	如果为 **true**，将 **/k8s/ns/active-namespace** 添加到起始位置。

console.navigation/resource-cluster
此扩展可用于贡献指向集群资源详情页面的导航项。该资源的 K8s 模型可用于定义导航项。

Name	值类型	选填	描述
id	**string**	否	此项目的唯一标识符。
model	**ExtensionK8sModel**	否	此导航项目所链接的模型。
perspective	**string**	是	此项目所属的视角 ID。若未指定，则向默认视角。
节	**string**	是	导航此项目所属的导航部分。如果未指定，请将此项目呈现为顶级链接。
dataAttributes	**{ [key: string]: string; }**	是	在 DOM 中添加数据属性。
startsWith	**string[]**	是	当 URL 从其中一个路径之一时，将此项标记为 active。
insertBefore	**string | string[]**	是	在此处引用的项目前插入此项。对于数组，使用按顺序找到的第一个。
insertAfter	**string | string[]**	是	在此处引用的项目后插入此项。对于数组，使用按顺序找到的第一个。**insertBefore** 具有高优先级。
name	**string**	是	覆盖默认名称。如果没有提供链接的名称，则会与模型的复数值相同。

console.navigation/resource-ns
此扩展可用于贡献指向命名空间资源详情页面的导航项。该资源的 K8s 模型可用于定义导航项。

Name	值类型	选填	描述
id	**string**	否	此项目的唯一标识符。
model	**ExtensionK8sModel**	否	此导航项目所链接的模型。
perspective	**string**	是	此项目所属的视角 ID。若未指定，则向默认视角。
节	**string**	是	导航此项目所属的导航部分。如果未指定，请将此项目呈现为顶级链接。
dataAttributes	**{ [key: string]: string; }**	是	在 DOM 中添加数据属性。
startsWith	**string[]**	是	当 URL 从其中一个路径之一时，将此项标记为 active。
insertBefore	**string | string[]**	是	在此处引用的项目前插入此项。对于数组，使用按顺序找到的第一个。
insertAfter	**string | string[]**	是	在此处引用的项目后插入此项。对于数组，使用按顺序找到的第一个。**insertBefore** 具有高优先级。
name	**string**	是	覆盖默认名称。如果没有提供链接的名称，则会与模型的复数值相同。

console.navigation/section
此扩展可用于在导航选项卡中定义导航项的新部分。

Name	值类型	选填	描述
id	**string**	否	此项目的唯一标识符。
perspective	**string**	是	此项目所属的视角 ID。若未指定，则向默认视角。

Name	值类型	选填	描述
dataAttributes	**{ [key: string]: string; }**	是	在 DOM 中添加数据属性。
insertBefore	**string | string[]**	是	在此处引用的项目前插入此项。对于数组，使用按顺序找到的第一个。
insertAfter	**string | string[]**	是	在此处引用的项目后插入此项。对于数组，使用按顺序找到的第一个。**insertBefore** 具有高优先级。
name	**string**	是	本节的名称。如果没有提供，上面部分将仅显示一个分隔符。

console.navigation/separator
此扩展可用于在导航中的导航项目之间添加分隔符。

Name	值类型	选填	描述
id	**string**	否	此项目的唯一标识符。
perspective	**string**	是	此项目所属的视角 ID。若未指定，则向默认视角。
节	**string**	是	导航此项目所属的导航部分。如果未指定，请将此项目呈现为顶级链接。
dataAttributes	**{ [key: string]: string; }**	是	在 DOM 中添加数据属性。
insertBefore	**string | string[]**	是	在此处引用的项目前插入此项。对于数组，使用按顺序找到的第一个。
insertAfter	**string | string[]**	是	在此处引用的项目后插入此项。对于数组，使用按顺序找到的第一个。**insertBefore** 具有高优先级。

console.page/resource/details

Name	值类型	选填	描述
model	**ExtensionK8sGroupKindModel**	否	此资源页面链接到的型号。
component	**CodeRef<React.ComponentType<{ match: match<{}>; namespace: string; model: ExtensionK8sModel; }>>**	否	路由匹配时要呈现的组件。

console.page/resource/list
向控制台路由器添加新的资源列表页面。

Name	值类型	选填	描述
model	**ExtensionK8sGroupKindModel**	否	此资源页面链接到的型号。
component	**CodeRef<React.ComponentType<{ match: match<{}>; namespace: string; model: ExtensionK8sModel; }>>**	否	路由匹配时要呈现的组件。

console.page/route
在 Web 控制台路由器中添加新页面。请参阅 React Router。

Name	值类型	选填	描述
component	**CodeRef<React.ComponentType<RouteComponentProps<{}, StaticContext, any>>>**	否	路由匹配时要呈现的组件。
path	**string | string[]**	否	**path-to-regexp@^1.7.0** 可以理解的有效 URL 路径或路径数组。
perspective	**string**	是	此页面所属的视图。如果没有指定，则会对所有视角贡献。

Name	值类型	选填	描述
exact	**布尔值**	是	为 true 时，只有在路径与 **location.pathname** 完全匹配时才会匹配。

console.page/route/standalone
向 web 控制台路由器中添加在通用页面布局外呈现的新独立页面。请参阅 React Router。

Name	值类型	选填	描述
component	**CodeRef<React.ComponentType<RouteComponentProps<{}, StaticContext, any>>>**	否	路由匹配时要呈现的组件。
path	**string | string[]**	否	**path-to-regexp@^1.7.0** 可以理解的有效 URL 路径或路径数组。
exact	**布尔值**	是	为 true 时，只有在路径与 **location.pathname** 完全匹配时才会匹配。

console.perspective
此扩展为控制台贡献一个新的视角，它允许自定义导航菜单。

Name	值类型	选填	描述
id	**string**	否	透视图标识符。
name	**string**	否	视角显示名称。
icon	**CodeRef<LazyComponent>**	否	视角显示图标。
landingPageURL	**CodeRef<(flags: { [key: string]: boolean; }, isFirstVisit: boolean) ⇒ string>**	否	获取视角登录页面 URL 的功能。
importRedirectURL	**CodeRef<(namespace: string) ⇒ string>**	否	获取导入流的重定向 URL。

Name	值类型	选填	描述
default	**布尔值**	是	这个视角是默认的。只能有一个默认值。
defaultPins	**ExtensionK8sModel[ ]**	是	nav 上的默认固定资源
usePerspectiveDetection	**CodeRef<() ⇒ [boolean, boolean]>**	是	要检测默认视角的 hook

console.project-overview/inventory-item
在 Project Overview 页面中添加一个新清单项。

Name	值类型	选填	描述
component	**CodeRef<React.ComponentType<{ projectName: string; }>>**	否	要呈现的组件。

console.project-overview/utilization-item
添加新项目概述使用率项目。

Name	值类型	选填	描述
title	**string**	否	使用项目的标题。
getUtilizationQuery	**CodeRef<GetProjectQuery>**	否	Prometheus 使用率查询。
humanize	**CodeRef<Humanize>**	否	将 Prometheus 数据转换为人类可读形式。
getTotalQuery	**CodeRef<GetProjectQuery>**	是	Prometheus 总计查询。
getRequestQuery	**CodeRef<GetProjectQuery>**	是	Prometheus 请求查询。
getLimitQuery	**CodeRef<GetProjectQuery>**	是	Prometheus 限制查询。
TopConsumerPopover	**CodeRef<React.ComponentType<TopConsumerPopoverProps>>**	是	显示 top consumer 弹出窗而不是纯值。

console.pvc/alert
此扩展可用于在 PVC 详情页面中贡献自定义警报。

Name	值类型	选填	描述
alert	**CodeRef<React.ComponentType<{ pvc: K8sResourceCommon; }>>**	否	警报组件。

console.pvc/create-prop
此扩展可以用来指定在 PVC 列表页面中创建 PVC 资源时使用的附加属性。

Name	值类型	选填	描述
label	**string**	否	创建 prop 操作的标签。
path	**string**	否	创建 prop 操作的路径。

console.pvc/delete
此扩展允许 hook 删除 PVC 资源。它可以为警报提供额外的信息和自定义 PVC 删除逻辑。

Name	值类型	选填	描述
predicate	**CodeRef<(pvc: K8sResourceCommon) ⇒ boolean>**	否	告知是否使用扩展名的 predicate。
onPVCKill	**CodeRef<(pvc: K8sResourceCommon) ⇒ Promise<void>>**	否	PVC 删除操作的方法。
alert	**CodeRef<React.ComponentType<{ pvc: K8sResourceCommon; }>>**	否	警报组件以显示其他信息。

console.pvc/status

Name	值类型	选填	描述
priority	**number**	否	状态组件的优先级。较大的值代表优先级更高。

Name	值类型	选填	描述
status	**CodeRef<React.ComponentType<{ pvc: K8sResourceCommon; }>>**	否	状态组件。
predicate	**CodeRef<(pvc: K8sResourceCommon) ⇒ boolean>**	否	指示是否呈现状态组件的 predicate。

console.redux-reducer
为 Console Red Hatux 存储添加了新的减少程序，该存储在 **plugins.<scope>** 子状态上运行。

Name	值类型	选填	描述
scope	**string**	否	代表 Red Hatux 状态对象中减少管理的子状态的关键。
reducer	**CodeRef<Reducer<any, AnyAction>>**	否	reducer 函数，在 reducer-managed substate 中操作。

console.resource/create
此扩展允许插件为特定资源提供自定义资源（如向导或表单）的自定义资源，当用户尝试创建新资源实例时，这些组件会被呈现。

Name	值类型	选填	描述
model	**ExtensionK8sModel**	否	此创建资源页面将呈现的型号
component	**CodeRef<React.ComponentType<CreateResourceComponentProps>>**	否	当模型匹配时要呈现的组件

console.resource/details-item
在详情页面的默认资源摘要中添加一个新的详情项。

Name	值类型	选填	描述
model	**ExtensionK8sModel**	否	主题资源的 API 组、版本和类型。
id	**string**	否	唯一标识符。

Name	值类型	选填	描述
column	**DetailsItemColumn**	否	确定项目是否将显示在详细信息页面上的资源摘要的"左"或"右"列中。默认："右"
title	**string**	否	详情项标题。
path	**string**	是	一个可选的、完全限定的、资源属性的路径，用作详情项目值。只能直接渲染 primitive type 值。使用 component 属性处理其他数据类型。
component	**CodeRef<React.ComponentType<DetailsItemComponentProps<K8sResourceCommon, any>>>**	是	可选的 React 组件，用于呈现详情项目值。
sortWeight	**number**	是	可选的排序权重，相对于同一列中的所有其他详细信息项目。由任何有效的 JavaScriptNumber 表示。每个列中的项目都被独立排序，从最低到最高。没有排序权重的项目在带有排序权重的项目排序。

console.storage-class/provisioner
在存储类创建过程中，添加新的存储类置备程序作为选项。

Name	值类型	选填	描述
CSI	**ProvisionerDetails**	是	Container Storage Interface provisioner 类型
OTHERS	**ProvisionerDetails**	是	其他置备程序类型

console.storage-provider
此扩展可用于生成新的存储供应商，以便在附加存储和供应商特定组件时进行选择。

Name	值类型	选填	描述
name	**string**	否	显示提供程序的名称。

Name	值类型	选填	描述
组件	**CodeRef<React.ComponentType<Partial<RouteComponentProps<{}, StaticContext, any>>>>**	否	要呈现的供应商特定组件。

console.tab
向与 **contextId** 匹配的 nav 中添加一个标签页。

Name	值类型	选填	描述
contextId	**string**	否	分配给水平 nav 的上下文 ID，在其中注入选项卡。可能的值： **dev-console-observe**
name	**string**	否	选项卡的显示标签
href	**string**	否	**href** 附加到现有 URL
component	**CodeRef<React.ComponentType<PageComponentProps<K8sResourceCommon>>>**	否	选项卡内容组件。

console.tab/horizontalNav
此扩展可用于在资源详情页面中添加标签页。

Name	值类型	选填	描述
model	**ExtensionK8sKindVersionModel**	否	此提供程序显示选项卡的型号。
page	**{ name: string; href: string; }**	否	要在水平标签页中显示的页面。它取标签名称作为名称，并且是 href 的选项卡
component	**CodeRef<React.ComponentType<PageComponentProps<K8sResourceCommon>>>**	否	路由匹配时要呈现的组件。

console.telemetry/listener
此组件可用于注册接收遥测事件的监听程序功能。这些事件包括用户识别、页面导航和其他特定应用程序的事件。侦听器可以使用这些数据进行报告和分析目的。

Name	值类型	选填	描述
listener	**CodeRef<TelemetryEventListener>**	否	侦听遥测事件

console.topology/adapter/build
BuildAdapter 贡献适配器将元素适应 Build 组件可以使用的数据

Name	值类型	选填	描述
adapt	**CodeRef<(element: GraphElement) ⇒ AdapterDataType<BuildConfigData> | undefined>**	否	用于调整元素以适用于 Build 组件的适配器。

console.topology/adapter/network
NetworkAdpater 贡献适配器，将元素适应数据，以供 **Networking** 组件使用

Name	值类型	选填	描述
adapt	**CodeRef<(element: GraphElement) ⇒ NetworkAdapterType | undefined>**	否	用于调整元素以适用于 Networking 组件的适配器。

console.topology/adapter/pod
PodAdapter 贡献一个适配器，将元素适应 **Pod** 组件可以使用的数据。

Name	值类型	选填	描述
adapt	**CodeRef<(element: GraphElement) ⇒ AdapterDataType<PodsAdapterDataType> | undefined>**	否	用于调整元素以适用于 Pod 组件的适配器。

console.topology/component/factory
ViewComponentFactory 的 getter。

Name	值类型	选填	描述

Name	值类型	选填	描述
getFactory	**CodeRef<ViewComponentFactory>**	否	**ViewComponentFactory** 的 getter。

console.topology/create/connector
创建连接器功能的 getter。

Name	值类型	选填	描述
getCreateConnector	**CodeRef<CreateConnectionGetter>**	否	创建连接器功能的 getter。

console.topology/data/factory
拓扑数据模型工厂扩展

Name	值类型	选填	描述
id	**string**	否	工厂的唯一 ID。
priority	**number**	否	工厂的优先级
资源	**WatchK8sResourcesGeneric**	是	要从 **useK8sWatchResources** hook 获取的资源。
workloadKeys	**string[]**	是	包含工作负载的资源中的密钥。
getDataModel	**CodeRef<TopologyDataModelGetter>**	是	数据模型工厂的 getter。
isResourceDepicted	**CodeRef<TopologyDataModelDepicted>**	是	用于确定资源是否由这个模型描述的 getter。
getDataModelReconciler	**CodeRef<TopologyDataModelReconciler>**	是	在所有扩展模型加载后用于协调数据模型的功能。

console.topology/decorator/provider
拓扑声明器供应商扩展

Name	值类型	选填	描述
id	**string**	否	拓扑 decorator 的 ID，特定于扩展

Name	值类型	选填	描述
priority	**number**	否	拓扑 decorator 的优先级，特定于扩展
quadrant	**TopologyQuadrant**	否	拓扑 decorator 的 Quadrant，特定于扩展
decorator	**CodeRef<TopologyDecoratorGetter>**	否	特定于扩展的 decorator

console.topology/details/resource-alert
DetailsResourceAlert 为特定拓扑上下文或图形元素贡献警报。

Name	值类型	选填	描述
id	**string**	否	此警报的 ID。用于保存状态，如果警报在丢弃后不应显示。
contentProvider	**CodeRef<(element: GraphElement) ⇒ DetailsResourceAlertContent | null>**	否	返回警报内容的 hook。

console.topology/details/resource-link
DetailsResourceLink 为特定拓扑上下文或图形元素贡献一个链接。

Name	值类型	选填	描述
link	**CodeRef<(element: GraphElement) ⇒ React.Component | undefined>**	否	如果提供，则返回资源链接，否则未定义。将 **ResourceIcon** 和 **ResourceLink** 属性用于风格。
priority	**number**	是	一个高的优先级可以在第一个机会出现时创建链接。

console.topology/details/tab
DetailsTab 为拓扑详情面板提供标签页。

Name	值类型	选填	描述
id	**string**	否	此详细信息标签的唯一标识符。

Name	值类型	选填	描述
label	**string**	否	要在 UI 中显示的标签标签。
insertBefore	**string | string[]**	是	在此处引用的项目前插入此项。对于数组，使用按顺序找到的第一个。
insertAfter	**string | string[]**	是	在此处引用的项目后插入此项。对于数组，使用按顺序找到的第一个。**insertBefore** 值具有优先权。

console.topology/details/tab-section
DetailsTabSection 对拓扑详情面板中的特定标签页贡献了一个部分。

Name	值类型	选填	描述
id	**string**	否	此详细信息选项卡部分的唯一标识符。
tab	**string**	否	本节应贡献的父选项卡 ID。
provider	**CodeRef<DetailsTabSectionExtensionHook>**	否	返回组件的 hook，或者如果为 null 或未定义，会在拓扑栏中显示。SDK 组件：**<Section title=\{}>...** padded 区域
节	**CodeRef<(element: GraphElement, renderNull?: () ⇒ null) ⇒ React.Component | undefined>**	否	Deprecated: 如果没有定义供应商会 Fallback，renderNull 已是一个 no-op。
insertBefore	**string | string[]**	是	在此处引用的项目前插入此项。对于数组，使用按顺序找到的第一个。
insertAfter	**string | string[]**	是	在此处引用的项目后插入此项。对于数组，使用按顺序找到的第一个。**insertBefore** 值具有优先权。

console.topology/display/filters

拓扑显示过滤器扩展

Name	值类型	选填	描述
getTopologyFilters	**CodeRef<() ⇒ TopologyDisplayOption[]>**	否	特定于扩展的拓扑过滤器的 getter
applyDisplayOptions	**CodeRef<TopologyApplyDisplayOptions>**	否	将过滤器应用到模型的功能

console.topology/relationship/provider
拓扑关系供应商连接器扩展

Name	值类型	选填	描述
provides	**CodeRef<RelationshipProviderProvides>**	否	用于确定是否可以在源和目标节点之间创建连接
工具提示	**string**	否	显示连接器操作将鼠标悬停在 drop 目标的工具，例如："Create a Visual Connector"
create	**CodeRef<RelationshipProviderCreate>**	否	连接器通过目标节点丢弃时执行的回调以创建连接
priority	**number**	否	关系的优先级，在有多个情况下，首选更高的优先级

console.user-preference/group
此扩展可用于在控制台 user-preferences 页面中添加组。它将在控制台 user-preferences 页面中显示为垂直标签页选项。

Name	值类型	选填	描述
id	**string**	否	用于识别用户首选项组的 ID。
label	**string**	否	用户首选项组的标签
insertBefore	**string**	是	应该放置此组前的用户首选项组群 ID
insertAfter	**string**	是	应该放置此组的用户首选项组群 ID

console.user-preference/item
此扩展可用于在控制台用户首选项页面中的用户首选项组群中添加项目。

Name	值类型	选填	描述
id	**string**	否	ID 用于识别用户首选项项目，并在 insertAfter 和 insertBefore 中引用来定义项目顺序
label	**string**	否	用户首选项的标签
description	**string**	否	用户首选项的描述
field	**UserPreferenceField**	否	用于呈现值来设置用户首选项的输入字段选项
groupId	**string**	是	用于识别项目所属的用户首选项组的 ID
insertBefore	**string**	是	应放置此项目前的用户首选项项目 ID
insertAfter	**string**	是	用户首选项项目的 ID，之后应放置此项目

console.yaml-template
通过 yaml 编辑器编辑资源的 YAML 模板。

Name	值类型	选填	描述
model	**ExtensionK8sModel**	否	与模板关联的模型。
模板	**CodeRef<string>**	否	YAML 模板。
name	**string**	否	模板的名称。使用名称 **default** 将其标记为默认模板。

dev-console.add/action
此扩展允许插件在开发人员视角的 add 页面中贡献 add action 项。例如，无服务器插件可以添加一个新的 action 项，用于在开发人员控制台的添加页面中添加无服务器功能。

Name	值类型	选填	描述
id	**string**	否	用于识别操作的 ID。

Name	值类型	选填	描述
label	**string**	否	操作的标签。
description	**string**	否	操作的描述。
href	**string**	否	要进入到的 **href**。
groupId	**string**	是	用于识别该操作所属的操作组的 ID。
icon	**CodeRef<React.ReactNode>**	是	视角显示图标。
accessReview	**AccessReviewResourceAttributes[]**	是	可选访问查看来控制操作的可见性或启用。

dev-console.add/action-group
此扩展允许插件在开发人员控制台的添加页面中继续组。组可以被操作引用，这些操作将根据它们的扩展定义在 add 操作页面中分组。例如，Serverless 插件可以贡献 Serverless 组，以及多个 add 操作。

Name	值类型	选填	描述
id	**string**	否	用于识别操作组的 ID
name	**string**	否	操作组的标题
insertBefore	**string**	是	应该放置此组的操作组群 ID
insertAfter	**string**	是	应该放置此组的操作组群 ID

dev-console.import/environment
此扩展可用于在开发人员控制台 git import 表单的构建器镜像选择器下指定额外的构建环境变量字段。设置后，字段将覆盖 build 部分中相同名称的环境变量。

Name	值类型	选填	描述
imageStreamName	**string**	否	为提供自定义环境变量的镜像流名称
imageStreamTags	**string[]**	否	支持的镜像流标签列表
environments	**ImageEnvironment[]**	否	环境变量列表

console.dashboards/overview/detail/item

弃用。使用 **CustomOverviewDetailItem** 类型

Name	值类型	选填	描述
component	**CodeRef<React.ComponentType<{}>>**	否	基于 **DetailItem** 组件的值

console.page/resource/tab
已弃用。使用 **console.tab/horizontalNav** 替代。添加新资源选项卡页面到控制台路由器。

Name	值类型	选填	描述
model	**ExtensionK8sGroupKindModel**	否	此资源页面链接到的型号。
component	**CodeRef<React.ComponentType<RouteComponentProps<{}, StaticContext, any>>>**	否	路由匹配时要呈现的组件。
name	**string**	否	选项卡的名称。
href	**string**	是	选项卡链接的可选 **href**。如果没有提供，则使用第一个 **path**。
exact	**布尔值**	是	为 true 时，只有在路径与 **location.pathname** 完全匹配时才会匹配。

7.5.2. 动态插件 API

useActivePerspective
提供用于设置活跃视角的 hook，以及用于设置活跃视角的回调。它返回一个包含当前活跃视角和 setter 回调的元组。

Example

```
const Component: React.FC = (props) => {
  const [activePerspective, setActivePerspective] = useActivePerspective();
  return <select
    value={activePerspective}
    onChange={(e) => setActivePerspective(e.target.value)}
  >
    {
      // ...perspective options
    }
  </select>
}
```

GreenCheckCircleIcon
显示绿色勾号图标的组件。

Example

```
<GreenCheckCircleIcon title="Healthy" />
```

参数名称	描述
className	（可选）组件的额外类名称
title	（可选）图标标题
size	（可选）图标大小：(**sm**,**md**,**lg**,**xl**)

RedExclamationCircleIcon
用于显示感叹号圆圈图标的组件。

Example

```
<RedExclamationCircleIcon title="Failed" />
```

参数名称	描述
className	（可选）组件的额外类名称
title	（可选）图标标题
size	（可选）图标大小：(**sm**,**md**,**lg**,**xl**)

YellowExclamationTriangleIcon
用于显示一个黄色三角形感叹号图标的组件。

Example

```
<YellowExclamationTriangleIcon title="Warning" />
```

参数名称	描述
className	（可选）组件的额外类名称
title	（可选）图标标题
size	（可选）图标大小：(**sm**,**md**,**lg**,**xl**)

BlueInfoCircleIcon

用于显示信息的蓝色圆圈图标的组件。

Example

```
<BlueInfoCircleIcon title="Info" />
```

参数名称	描述
className	（可选）组件的额外类名称
title	（可选）图标标题
size	（可选）图标大小：('sm', 'md', 'lg', 'xl')

ErrorStatus
用于显示错误状态弹出的组件。

Example

```
<ErrorStatus title={errorMsg} />
```

参数名称	描述
title	（可选）状态文本
iconOnly	（可选）如果为 true，则仅显示图标
noTooltip	（可选）如果为 true，则不会显示工具提示
className	（可选）组件的额外类名称
popoverTitle	（可选）弹出的标题

InfoStatus
用于显示信息状态弹出窗口的组件。

Example

```
<InfoStatus title={infoMsg} />
```

参数名称	描述
title	（可选）状态文本
iconOnly	（可选）如果为 true，则仅显示图标

参数名称	描述
noTooltip	（可选）如果为 true，则不会显示工具提示
className	（可选）组件的额外类名称
popoverTitle	（可选）弹出的标题

ProgressStatus
显示进度状态弹出的组件。

Example

```
<ProgressStatus title={progressMsg} />
```

参数名称	描述
title	（可选）状态文本
iconOnly	（可选）如果为 true，则仅显示图标
noTooltip	（可选）如果为 true，则不会显示工具提示
className	（可选）组件的额外类名称
popoverTitle	（可选）弹出的标题

SuccessStatus
用于显示成功状态弹出窗口的组件。

Example

```
<SuccessStatus title={successMsg} />
```

参数名称	描述
title	（可选）状态文本
iconOnly	（可选）如果为 true，则仅显示图标
noTooltip	（可选）如果为 true，则不会显示工具提示
className	（可选）组件的额外类名称
popoverTitle	（可选）弹出的标题

checkAccess
提供有关用户对给定资源的访问权限的信息。它返回一个具有资源访问信息的对象。

参数名称	描述
resourceAttributes	访问查看的资源属性
impersonate	模拟详情

useAccessReview
hook，提供有关用户对给定资源的访问权限的信息。它返回一个带有 **isAllowed** 和 **loading** 值的数组。

参数名称	描述
resourceAttributes	访问查看的资源属性
impersonate	模拟详情

useResolvedExtensions
使用已解析的 **CodeRef** 属性为消耗控制台扩展做出反应 hook。此 hook 接受与 **useExtensions** hook 相同的参数，并返回自适应扩展实例列表，从而解析每个扩展属性中的所有代码引用。

最初，hook 返回一个空数组。完成解析后，React 组件将重新渲染，使用 hook 返回已修改的扩展列表。当匹配扩展列表更改时，会重启解析。hook 继续返回前面的结果，直到解析完成为止。

hook 结果元素保证在重新方之间保持稳定。它返回一个包含已解析代码引用的自适应扩展实例列表的元组、指示解析是否完成的布尔值标志，以及在解析过程中检测到的错误列表。

Example

```
const [navItemExtensions, navItemsResolved] = useResolvedExtensions<NavItem>(isNavItem);
// process adapted extensions and render your component
```

参数名称	描述
typeGuards	每个接受动态插件扩展作为参数的回调列表，并返回布尔值标记，指示扩展是否满足所需的类型限制

HorizontalNav
为页面创建导航栏的组件。路由作为组件的一部分进行处理。**console.tab/horizontalNav** 可用于向任何横向导航中添加额外的内容。

Example

```
const HomePage: React.FC = (props) => {
  const page = {
    href: '/home',
    name: 'Home',
    component: () => <>Home</>
```

```
    }
    return <HorizontalNav match={props.match} pages={[page]} />
}
```

参数名称	描述
resource	与这个 Navigation 关联的资源，对象为 K8sResourceCommon 类型
pages	页面对象数组
match	匹配 React Router 提供的对象

VirtualizedTable
创建虚拟化表的组件。

Example

```
const MachineList: React.FC<MachineListProps> = (props) => {
  return (
    <VirtualizedTable<MachineKind>
     {...props}
     aria-label='Machines'
     columns={getMachineColumns}
     Row={getMachineTableRow}
    />
  );
}
```

参数名称	描述
data	表的数据
loaded	表示数据已加载的标记
loadError	如果问题载入数据，则错误对象
columns	列设置
Row	行设置
unfilteredData	没有过滤器的原始数据
NoDataEmptyMsg	（可选）没有数据空消息组件
EmptyMsg	（可选）空消息组件
scrollNode	（可选）处理滚动的功能

参数名称	描述
label	（可选）表的标签
ariaLabel	（可选）aria 标签
gridBreakPoint	调整用于响应的网格如何分隔
onSelect	（可选）处理表选择的功能
rowData	（可选）特定于行的数据

TableData
用于在表行中显示表数据的组件。

Example

```
const PodRow: React.FC<RowProps<K8sResourceCommon>> = ({ obj, activeColumnIDs }) => {
  return (
    <>
      <TableData id={columns[0].id} activeColumnIDs={activeColumnIDs}>
        <ResourceLink kind="Pod" name={obj.metadata.name} namespace={obj.metadata.namespace}
/>
      </TableData>
      <TableData id={columns[1].id} activeColumnIDs={activeColumnIDs}>
        <ResourceLink kind="Namespace" name={obj.metadata.namespace} />
      </TableData>
    </>
  );
};
```

参数名称	描述
id	表的唯一 ID
activeColumnIDs	active 列
className	（可选） 风格的选项类名称

useActiveColumns
一个 hook，它提供了用户选择的 active TableColumns 列表。

Example

```
// See implementation for more details on TableColumn type
  const [activeColumns, userSettingsLoaded] = useActiveColumns({
    columns,
    showNamespaceOverride: false,
```

```
  columnManagementID,
});
return userSettingsAreLoaded ? <VirtualizedTable columns={activeColumns} {...otherProps} /> : null
```

参数名称	描述
options	哪些作为键-值映射传递
\{TableColumn[]} options.columns	所有可用 TableColumns 的数组
{boolean} [options.showNamespaceOverride]	（可选）如果为 true，则会包含命名空间列，无论列管理选择是什么
{string} [options.columnManagementID]	（可选）用于保留和检索列管理选择的唯一 ID，并从用户设置中保留和检索列管理选择。通常，一个资源的 group/version/kind (GVK) 字符串。

包含当前用户所选活跃列(options.columns 的子集)和布尔值标志，指示用户设置是否已加载。

ListPageHeader
用于生成页面标头的组件。

Example

```
const exampleList: React.FC = () => {
  return (
    <>
      <ListPageHeader title="Example List Page"/>
    </>
  );
};
```

参数名称	描述
title	头标题
helpText	（可选） help 部分作为响应节点
badge	（可选）作为响应节点的徽标图标

ListPageCreate
用于为特定资源类型添加创建按钮的组件，它们会自动为该资源创建 YAML 生成链接。

Example

```
const exampleList: React.FC<MyProps> = () => {
  return (
    <>
      <ListPageHeader title="Example Pod List Page"/>
```

```
      <ListPageCreate groupVersionKind="Pod">Create Pod</ListPageCreate>
    </ListPageHeader>
  </>
 );
};
```

参数名称	描述
groupVersionKind	要代表的资源组/版本/类型

ListPageCreateLink
用于创建样式链接的组件。

Example

```
const exampleList: React.FC<MyProps> = () => {
 return (
  <>
   <ListPageHeader title="Example Pod List Page"/>
     <ListPageCreateLink to={'/link/to/my/page'}>Create Item</ListPageCreateLink>
   </ListPageHeader>
  </>
 );
};
```

参数名称	描述
至	链接应连接到的字符串位置
createAccessReview	（可选）用于决定访问的命名空间和 kind 的对象
children	（可选）组件的子项

ListPageCreateButton
创建按钮的组件。

Example

```
const exampleList: React.FC<MyProps> = () => {
  return (
    <>
      <ListPageHeader title="Example Pod List Page"/>
        <ListPageCreateButton createAccessReview={access}>Create Pod</ListPageCreateButton>
      </ListPageHeader>
    </>
  );
};
```

参数名称	描述
createAccessReview	（可选）用于决定访问的命名空间和 kind 的对象
pfButtonProps	（可选） Patternfly Button props

ListPageCreateDropdown
创建权限检查嵌套的下拉菜单的组件。

Example

```
const exampleList: React.FC<MyProps> = () => {
  const items = {
    SAVE: 'Save',
    DELETE: 'Delete',
  }
  return (
    <>
     <ListPageHeader title="Example Pod List Page"/>
       <ListPageCreateDropdown createAccessReview={access}
items={items}>Actions</ListPageCreateDropdown>
     </ListPageHeader>
    </>
  );
};
```

参数名称	描述
items	Key:ReactNode 对，以便在组件下拉列表中显示
onClick	点下拉菜单项目的回调功能
createAccessReview	（可选）用于决定访问的命名空间和 kind 的对象
children	（可选）下拉菜单的子项

ListPageFilter
为列表页面生成过滤器的组件。

Example

```
  // See implementation for more details on RowFilter and FilterValue types
  const [staticData, filteredData, onFilterChange] = useListPageFilter(
    data,
    rowFilters,
    staticFilters,
  );
  // ListPageFilter updates filter state based on user interaction and resulting filtered data can be
rendered in an independent component.
  return (
```

```
    <>
      <ListPageHeader .../>
      <ListPagBody>
        <ListPageFilter data={staticData} onFilterChange={onFilterChange} />
        <List data={filteredData} />
      </ListPageBody>
    </>
  )
```

参数名称	描述
data	一个数据点的数组
loaded	表示数据已加载
onFilterChange	更新过滤器时的回调功能
rowFilters	（可选）定义可用过滤器选项的 RowFilter 元素的数组
nameFilterPlaceholder	（可选）名称过滤器的占位符
labelFilterPlaceholder	（可选）标签过滤器的占位符
hideLabelFilter	（可选）仅显示 name 过滤器，而不是 name 和 label 过滤器
hideNameLabelFilter	（可选）隐藏 name 和 label filter
columnLayout	（可选）列布局对象
hideColumnManagement	（可选）隐藏列管理的标记

useListPageFilter
管理 ListPageFilter 组件的过滤器状态的 hook。它返回一个 tuple，其中包含由所有静态过滤器过滤的数据、由所有静态和行过滤器过滤的数据，以及更新 rowFilters 的回调。

Example

```
  // See implementation for more details on RowFilter and FilterValue types
  const [staticData, filteredData, onFilterChange] = useListPageFilter(
    data,
    rowFilters,
    staticFilters,
  );
  // ListPageFilter updates filter state based on user interaction and resulting filtered data can be
rendered in an independent component.
  return (
    <>
      <ListPageHeader .../>
      <ListPagBody>
```

```
      <ListPageFilter data={staticData} onFilterChange={onFilterChange} />
      <List data={filteredData} />
    </ListPageBody>
  </>
)
```

参数名称	描述
data	一个数据点的数组
rowFilters	（可选）定义可用过滤器选项的 RowFilter 元素的数组
staticFilters	（可选）静态应用到数据的 FilterValue 元素的数组

ResourceLink
创建指向带有图标徽标的特定资源类型的链接的组件。

Example

```
<ResourceLink
  kind="Pod"
  name="testPod"
  title={metadata.uid}
/>
```

参数名称	描述
kind	（可选）资源（如 Pod、Deployment、Namespace）的类型
groupVersionKind	（可选）带有 group、version 和 kind 的对象
className	（可选）组件类风格
displayName	（可选）组件显示名称，如果设置则覆盖资源名称
inline	（可选）使用子项创建图标徽标和名称标记
linkTo	（可选）创建 Link 对象的标记 - 默认为 true
name	（可选）资源名称
namesapce	（可选）要链接到的 kind 资源的特定命名空间
hideIcon	（可选）隐藏图标徽标的标记
title	（可选）链接对象的标题（未显示）

参数名称	描述
dataTest	（可选）用于测试的标识符
onClick	（可选）当点组件时的回调功能
truncate	（可选）用于截断链接（如果太长）的标记

ResourceIcon
为特定资源类型创建图标徽标的组件。

Example

```
<ResourceIcon kind="Pod"/>
```

参数名称	描述
kind	（可选）资源（如 Pod、Deployment、Namespace）的类型
groupVersionKind	（可选）带有 group、version 和 kind 的对象
className	（可选）组件类风格

useK8sModel
用于为来自 redux 的 K8sGroupVersionKind 获取 k8s 模型的 hook。它返回一个数组，第一个项为 k8s 模型，第二个项目为 **inFlight** 状态。

Example

```
const Component: React.FC = () => {
  const [model, inFlight] = useK8sModel({ group: 'app'; version: 'v1'; kind: 'Deployment' });
  return ...
}
```

参数名称	描述
groupVersionKind	group, version, k8s 资源 kind K8sGroupVersionKind 是首选的，或为 group, version 传递引用，kind 已弃用，例如 group/version/kind (GVK) K8sResourceKindReference。

useK8sModels
从 redux 检索所有当前 k8s 模型的 hook。它返回一个数组，第一个项是 k8s 模型列表，第二个项目是 **inFlight** 状态。

Example

```
const Component: React.FC = () => {
  const [models, inFlight] = UseK8sModels();
  return ...
}
```

useK8sWatchResource
检索 k8s 资源以及加载和错误的状态的 hook。它返回一个数组，包含第一个项目作为资源，第二个项目作为加载的状态，第三个项目则作为错误状态（若有）。

Example

```
const Component: React.FC = () => {
  const watchRes = {
        ...
      }
  const [data, loaded, error] = useK8sWatchResource(watchRes)
  return ...
}
```

参数名称	描述
initResource	监视资源所需的选项。

useK8sWatchResources
检索 k8s 资源的 hook，以及它们的相应状态用于加载和错误。它返回一个映射，在 initResouces 中提供的键，值有三个属性 data, loaded 和 error。

Example

```
const Component: React.FC = () => {
  const watchResources = {
        'deployment': {...},
        'pod': {...}
        ...
      }
  const {deployment, pod} = useK8sWatchResources(watchResources)
  return ...
}
```

参数名称	描述
initResources	资源必须被监视为键值对，其中键对于资源是唯一的，值是监视相应资源所需的选项。

consoleFetch
一个围绕 **fetch** 添加控制台特定标头的自定义打包程序，并允许重试和超时。它还验证响应状态代码，并抛出适当的错误或根据需要注销用户。它返回一个可以解析到响应的承诺。

参数名称	描述
url	要获取的 URL
options	传递给获取的选项
timeout	以毫秒为单位的超时时间

consoleFetchJSON
围绕 **fetch** 的自定义打包程序，添加特定于控制台的标头并允许重试和超时。它还验证响应状态代码，并在需要时抛出适当的错误或注销用户。它将响应返回为 JSON 对象。在内部使用 **consoleFetch**。它返回一个承诺，它将解析为 JSON 对象的响应。

参数名称	描述
url	要获取的 URL
方法	要使用的 HTTP 方法。默认为 GET
options	传递给获取的选项
timeout	以毫秒为单位的超时时间
cluster	发出请求的集群名称。默认为用户选择的活跃集群

consoleFetchText
围绕 **fetch** 的自定义打包程序，添加特定于控制台的标头并允许重试和超时。它还验证响应状态代码，并在需要时抛出适当的错误或注销用户。它将响应返回为文本。在内部使用 **consoleFetch**。它返回一个能以文本方式解析到响应的承诺。

参数名称	描述
url	要获取的 URL
options	传递给获取的选项
timeout	以毫秒为单位的超时时间
cluster	发出请求的集群名称。默认为用户选择的活跃集群

getConsoleRequestHeaders
使用当前 redux 状态的 API 请求创建模拟和多集群相关标头的功能。它根据 redux 状态返回一个包含适当模拟和冲突的对象。

参数名称	描述
targetCluster	使用提供的 targetCluster 覆盖当前活跃的集群

k8sGetResource
它根据提供的选项从集群获取资源。如果提供了名称，它会返回一个资源，它会返回与模型匹配的所有资源。如果名称被提供，它会返回一个承诺，它将作为 JSON 对象解析到 JSON 对象（如果其返回与模型匹配的所有资源）。如果出现故障，则承诺将被拒绝，并附带 HTTP 错误响应。

参数名称	描述
options	在映射中作为键值对传递
options.model	k8s 模型
options.name	如果不提供资源名称，它会查找与模型匹配的所有资源。
options.ns	要查找的命名空间，不应为集群范围的资源指定。
options.path	如果提供，请附加为子路径
options.queryParams	URL 中包含的查询参数。
options.requestInit	要使用的 fetch init 对象。这可以具有请求标头、方法、重定向等。如需更多信息，请参阅 Interface RequestInit。

k8sCreateResource
它会根据提供的选项在集群中创建资源。它返回一个能够解析所创建的资源响应的承诺。如果失败承诺，则拒绝 HTTP 错误响应。

参数名称	描述
options	在映射中作为键值对传递
options.model	k8s 模型
options.data	要创建的资源的有效负载
options.path	如果提供，请附加为子路径
options.queryParams	URL 中包含的查询参数。

k8sUpdateResource

它根据提供的选项更新集群中的整个资源。当客户端需要完全替换现有资源时，它们可以使用 k8sUpdate。或者，也可以使用 k8sPatch 执行部分更新。它返回一个可以解析到更新的资源响应的承诺。如果失败承诺，则拒绝 HTTP 错误响应。

参数名称	描述
options	在映射中作为键值对传递
options.model	k8s 模型
options.data	要更新的 k8s 资源的有效负载
options.ns	要查找的命名空间，不应为集群范围的资源指定。
options.name	要更新的资源名称。
options.path	如果提供，请附加为子路径
options.queryParams	URL 中包含的查询参数。

k8sPatchResource
它根据提供的选项对集群中的任何资源进行补丁。当客户端需要执行部分更新时，可以使用 k8sPatch。或者可以使用 k8sUpdate 来完全替换现有资源。如需更多信息，请参阅数据跟踪器。它返回一个可以解析到修补资源响应的承诺。如果失败承诺，则拒绝 HTTP 错误响应。

参数名称	描述
options	在映射中作为键值对传递。
options.model	k8s 模型
options.resource	要修补的资源。
options.data	只有现有资源使用操作、路径和值修补的数据。
options.path	如果提供，请附加为子路径。
options.queryParams	URL 中包含的查询参数。

k8sDeleteResource
它根据提供的模型资源从集群中删除资源。垃圾回收基于 **Foreground|Background**，可以在提供的模型中使用 propagationPolicy 属性进行配置，或者在 json 中传递。它返回一个解析为 Status 的响应的承诺。如果失败承诺，则拒绝 HTTP 错误响应。

Example

kind: 'DeleteOptions', apiVersion: 'v1', propagationPolicy

参数名称	描述
options	在映射中作为键值对传递。
options.model	k8s 模型
options.resource	要删除的资源。
options.path	如果提供，请附加为子路径
options.queryParams	URL 中包含的查询参数。
options.requestInit	要使用的 fetch init 对象。这可以具有请求标头、方法、重定向等。如需更多信息，请参阅 Interface RequestInit。
options.json	如果其他提供，可以明确控制资源的垃圾回收，或者默认为模型的"propagationPolicy"。

k8sListResource
根据提供的选项，将资源列为集群中的数组。它返回一个可以解析到响应的承诺。

参数名称	描述
options	在映射中作为键值对传递
options.model	k8s 模型
options.queryParams	URL 中包含的查询参数，并可使用键 "labelSelector" 传递标签选择器。
options.requestInit	要使用的 fetch init 对象。这可以具有请求标头、方法、重定向等。如需更多信息，请参阅 Interface RequestInit。

k8sListResourceItems
与 k8sListResource 相同的接口，但会返回子项目。它返回模型的 apiVersion，即 **group/version**。

getAPIVersionForModel
为 k8s 模型提供 apiVersion。

参数名称	描述
model	k8s 模型

getGroupVersionKindForResource

为资源提供组、版本和类型。它返回提供的资源的组 version 和 kind。如果资源没有 API 组，则返回组"core"。如果资源具有无效的 apiVersion，它会抛出一个 Error。

参数名称	描述
resource	k8s 资源

getGroupVersionKindForModel
为 k8s 模型提供组、版本和类型。这会返回提供的模型的组版本 kind。如果模型没有 apiGroup，则返回组 "core"。

参数名称	描述
model	k8s 模型

StatusPopupSection
在弹出窗口中显示状态的组件。用于构建 **console.dashboards/overview/health/resource** 扩展的有用组件。

Example

```
<StatusPopupSection
  firstColumn={
    <>
      <span>{title}</span>
      <span className="text-secondary">
        My Example Item
      </span>
    </>
  }
  secondColumn='Status'
>
```

参数名称	描述
firstColumn	弹出的第一列的值
secondColumn	（可选）弹出的第二列的值
children	（可选）弹出窗口的子项

StatusPopupItem
状态弹出窗口中使用的 status 元素；在 **StatusPopupSection** 中使用。

Example

```
<StatusPopupSection
  firstColumn='Example'
  secondColumn='Status'
```

```
>
  <StatusPopupItem icon={healthStateMapping[MCGMetrics.state]?.icon}>
    Complete
  </StatusPopupItem>
  <StatusPopupItem icon={healthStateMapping[RGWMetrics.state]?.icon}>
     Pending
  </StatusPopupItem>
</StatusPopupSection>
```

参数名称	描述
value	（可选）要显示的文本值
icon	（可选）要显示的图标
children	子元素

概述
为仪表板创建一个打包程序组件。

Example

```
<Overview>
  <OverviewGrid mainCards={mainCards} leftCards={leftCards} rightCards={rightCards} />
</Overview>
```

参数名称	描述
className	（可选） div 的风格类
children	（可选）仪表板的元素

OverviewGrid
为仪表板创建卡元素的网格；在 **Overview** 中使用。

Example

```
<Overview>
  <OverviewGrid mainCards={mainCards} leftCards={leftCards} rightCards={rightCards} />
</Overview>
```

参数名称	描述
mainCards	网格的卡
leftCards	（可选）网格左侧的卡

参数名称	描述
rightCards	（可选）网格右侧的卡

InventoryItem
创建清单卡项。

Example

```
return (
  <InventoryItem>
    <InventoryItemTitle>{title}</InventoryItemTitle>
    <InventoryItemBody error={loadError}>
      {loaded && <InventoryItemStatus count={workerNodes.length} icon={<MonitoringIcon />} />}
    </InventoryItemBody>
  </InventoryItem>
)
```

参数名称	描述
children	在项目内呈现的元素

InventoryItemTitle
为清单卡项目创建标题；在 **InventoryItem** 内使用。

Example

```
return (
  <InventoryItem>
    <InventoryItemTitle>{title}</InventoryItemTitle>
    <InventoryItemBody error={loadError}>
      {loaded && <InventoryItemStatus count={workerNodes.length} icon={<MonitoringIcon />} />}
    </InventoryItemBody>
  </InventoryItem>
)
```

参数名称	描述
children	在标题内呈现的元素

InventoryItemBody
创建清单卡的正文；在 **InventoryCard** 中使用，并可与 **InventoryTitle** 一起使用。

Example

```
return (
  <InventoryItem>
    <InventoryItemTitle>{title}</InventoryItemTitle>
    <InventoryItemBody error={loadError}>
```

```
      {loaded && <InventoryItemStatus count={workerNodes.length} icon={<MonitoringIcon />} />}
    </InventoryItemBody>
  </InventoryItem>
 )
```

参数名称	描述
children	在清单卡或标题内呈现的元素
错误	div 的元素

InventoryItemStatus
为带有可选链接地址的清单卡创建一个计数和图标；在 **InventoryItemBody** 中使用

Example

```
 return (
  <InventoryItem>
   <InventoryItemTitle>{title}</InventoryItemTitle>
   <InventoryItemBody error={loadError}>
    {loaded && <InventoryItemStatus count={workerNodes.length} icon={<MonitoringIcon />} />}
   </InventoryItemBody>
  </InventoryItem>
 )
```

参数名称	描述
æ•°é‡	显示计数
icon	显示图标
linkTo	（可选）链接地址

InventoryItemLoading
为清单卡加载时创建框架容器；与 **InventoryItem** 和相关组件一起使用

Example

```
if (loadError) {
   title = <Link to={workerNodesLink}>{t('Worker Nodes')}</Link>;
} else if (!loaded) {
  title = <><InventoryItemLoading /><Link to={workerNodesLink}>{t('Worker Nodes')}</Link></>;
}
return (
  <InventoryItem>
    <InventoryItemTitle>{title}</InventoryItemTitle>
  </InventoryItem>
)
```

useFlag

从 FLAGS redux 状态返回给定功能标记的 hook。它返回请求的功能标记或未定义的布尔值。

参数名称	描述
flag	要返回的功能标志

CodeEditor
一个基本的 lazy 加载的代码编辑器，带有悬停的帮助和完成。

Example

```
<React.Suspense fallback={<LoadingBox />}>
  <CodeEditor
    value={code}
    language="yaml"
  />
</React.Suspense>
```

参数名称	描述
value	代表要呈现的 yaml 代码的字符串。
language	代表编辑器语言的字符串。
options	Monaco 编辑器选项.如需更多详细信息，请访问 Interface IStandAloneEditorConstructionOptions。
minHeight	有效 CSS 高度中的值的最小编辑器高度。
showShortcuts	布尔值，以显示编辑器顶部的快捷方式。
toolbarLinks	编辑器顶部的工具栏链接部分上 ReactNode rendered 的数组。
onChange	代码更改事件的回调。
onSave	触发命令 CTRL / CMD + S 时调用的回调。
ref	对 **{ editor?: IStandaloneCodeEditor }** 的响应参考。使用 **editor** 属性，您可以访问所有方法来控制编辑器。如需更多信息，请访问 Interface IStandaloneCodeEditor。

ResourceYAMLEditor
一个 lazy 载入的 Kubernetes 资源的 YAML 编辑器，带有悬停的帮助和完成。组件使用 YAMLEditor 并在其之上添加更多功能，如资源更新处理、警报、保存、取消和重新加载按钮、可访问性等。除非提供了 **onSave** 回调，否则自动处理资源更新。它应该嵌套在 **React.Suspense** 组件中。

Example

```
<React.Suspense fallback={<LoadingBox />}>
  <ResourceYAMLEditor
    initialResource={resource}
    header="Create resource"
    onSave={(content) => updateResource(content)}
  />
</React.Suspense>
```

参数名称	描述
initialResource	代表编辑器显示资源的 YAML/Object。这个 prop 仅在初始呈现过程中使用
header	在 YAML 编辑器之上添加一个标头
onSave	保存按钮的回调。传递它会覆盖编辑器对资源执行的默认更新

ResourceEventStream
显示与特定资源相关的事件的组件。

Example

```
const [resource, loaded, loadError] = useK8sWatchResource(clusterResource);
return <ResourceEventStream resource={resource} />
```

参数名称	描述
resource	应该会显示相关事件的对象。

usePrometheusPoll
为单个查询设置对 Prometheus 的轮询。它返回一个包含查询响应的元组、指示响应是否已完成的布尔值标志，以及请求请求或后处理过程中遇到的任何错误。

参数名称	描述
{PrometheusEndpoint} props.endpoint	PrometheusEndpoint 中的一个(label, query, range, rules, targets)
{string} [props.query]	（可选）Prometheus 查询字符串。如果为空或未定义，则不会启动轮询。
{number} [props.delay]	（可选）轮询延迟间隔 (ms)
{number} [props.endTime]	（可选）用于 QUERY_RANGE enpoint，查询范围的末尾

参数名称	描述
{number} [props.samples]	（可选）用于 QUERY_RANGE enpoint
{number} [options.timespan]	（可选）用于 QUERY_RANGE enpoint
{string} [options.namespace]	（可选）要附加的搜索参数
{string} [options.timeout]	（可选）要附加的搜索参数

Timestamp
呈现时间戳的组件。时间戳在 Timestamp 组件的 invidual 实例之间同步。提供的时间戳会根据用户区域设置进行格式化。

参数名称	描述
timestamp	呈现的时间戳。格式预期为 ISO 8601 （由 Kubernetes 使用）、epo 时间时间戳或日期的实例。
simple	呈现组件的简单版本，省略图标和工具提示。
omitSuffix	格式化日期消息后缀。
className	组件的额外类名称。

useModal
用于启动修改的 hook。

Example

```
const context: AppPage: React.FC = () => {<br/> const [launchModal] = useModal();<br/> const onClick = () => launchModal(ModalComponent);<br/> return (<br/>   <Button onClick={onClick}>Launch a Modal</Button><br/> )<br/>}<br/>`
```

ActionServiceProvider
允许从 **console.action/provider** 扩展类型的其他插件接收贡献的组件。

Example

```
  const context: ActionContext = { 'a-context-id': { dataFromDynamicPlugin } };

  ...

  <ActionServiceProvider context={context}>
    {({ actions, options, loaded }) =>
      loaded && (
        <ActionMenu actions={actions} options={options} variant={ActionMenuVariant.DROPDOWN}
/>
```

```
      )
    }
</ActionServiceProvider>
```

参数名称	描述
context	带有 contextId 和可选插件数据的对象

NamespaceBar
一个组件，它呈现一个水平工具栏，其中有一个命名空间下拉菜单（在最左边）。其他组件可以作为子项传递，并呈现到命名空间下拉菜单的右侧。此组件设计为在页面的顶部使用。它应用于用户需要更改活动命名空间的页面，如 k8s 资源的页面中。

Example

```
const logNamespaceChange = (namespace) => console.log(`New namespace: ${namespace}`);

...

<NamespaceBar onNamespaceChange={logNamespaceChange}>
  <NamespaceBarApplicationSelector />
</NamespaceBar>
<Page>

  ...
```

参数名称	描述
onNamespaceChange	（可选）当选择命名空间选项时执行的功能。它接受字符串形式的新命名空间作为其唯一参数。选择了选项时，活跃命名空间会自动更新，但可通过此功能应用其他逻辑。当更改命名空间时，URL 中的 namespace 参数会从以前的命名空间改为新选择的命名空间。
isDisabled	（可选）如果设为 true，则禁用命名空间下拉菜单的布尔值标志。这个选项只适用于命名空间下拉菜单，对子组件没有影响。
children	（可选）在命名空间下拉菜单右侧的工具栏中呈现的其他元素。

ErrorBoundaryFallbackPage
创建全页 ErrorBoundaryFallbackPage 组件以显示 "Oh no!Something went wrong." 消息与堆栈跟踪和其他有帮助的调试信息一起。这与组件一起使用。

Example

```
//in ErrorBoundary component
 return (
```

```
  if (this.state.hasError) {
    return <ErrorBoundaryFallbackPage errorMessage={errorString} componentStack=
{componentStackString}
      stack={stackTraceString} title={errorString}/>;
  }

  return this.props.children;
)
```

参数名称	描述
errorMessage	错误消息的文本描述
componentStack	异常的组件追踪
queue	异常的堆栈追踪
title	将标题显示为错误边界页面的标头

QueryBrowser
一个组件，它呈现来自 Prometheus PromQL 查询的结果图，以及用于与图形交互的控制。

Example

```
<QueryBrowser
  defaultTimespan={15 * 60 * 1000}
  namespace={namespace}
  pollInterval={30 * 1000}
  queries={[
    'process_resident_memory_bytes{job="console"}',
    'sum(irate(container_network_receive_bytes_total[6h:5m])) by (pod)',
  ]}
/>
```

参数名称	描述
customDataSource	（可选）处理 PromQL 查询的 API 端点的基本 URL。如果提供，则使用此选项而不是默认的 API 获取数据。
defaultSamples	（可选）每个数据系列绘制的默认数据样本数量。如果有多个数据系列，QueryBrowser 可能会自动选择比此处指定的数据样本的数量较低。
defaultTimespan	（可选）图形的默认时间跨度（以毫秒为单位） - 默认为 1,800,000 (30 分钟)。

参数名称	描述
disabledSeries	（可选） 禁用（不显示）带有这些准确标签/值对的数据系列。
disableZoom	（可选）禁用图形缩放控制的标记。
filterLabels	（可选）将返回的数据系列过滤到仅与这些标签/值对匹配的数据系列。
fixedEndTime	（可选）为显示的时间范围设置结束时间，而不是显示到当前时间的数据。
formatSeriesTitle	（可选）返回字符串的功能，用作单个数据系列的标题。
GraphLink	（可选）呈现到另一个页面的链接（例如，获取有关此查询的更多信息）。
hideControls	（可选）用于隐藏用于更改图形 timespan 的图形控制的标记，等等。
isStack	（可选） 显示堆栈图而不是行图的标记。如果设置了 showStackedControl，用户仍可切换到行图。
namespace	（可选）如果提供，则只为这个命名空间返回数据（只有具有此命名空间标签的系列）。
onZoom	（可选）当图形缩放时调用的回调。
pollInterval	（可选）如果设置，请确定图形更新的频率，以显示最新数据（以毫秒为单位）。
queries	运行 PromQL 查询的数组，并显示图表中的结果。
showLegend	（可选）启用在图形下显示图下的标记。
showStackedControl	启用显示图形模式和行图形模式切换的图形控制的标志。
timespan	（可选）图形应涵盖的时间持续时间（以毫秒为单位）。
units	（可选）要在 Y 轴和工具提示中显示的单元。

useAnnotationsModal
提供回调来启动用于编辑 Kubernetes 资源注解的模态的 hook。

Example

```
const PodAnnotationsButton = ({ pod }) => {
  const { t } = useTranslation();
  const launchAnnotationsModal = useAnnotationsModal<PodKind>(pod);
  return <button onClick={launchAnnotationsModal}>{t('Edit Pod Annotations')}</button>
}
```

参数名称	描述
resource	用于编辑 K8sResourceCommon 类型的对象的注解的资源。

返回

启动一个模态来编辑资源注解的功能。

useDeleteModal
提供回调用于启动删除资源的模态的 hook。

Example

```
const DeletePodButton = ({ pod }) => {
  const { t } = useTranslation();
  const launchDeleteModal = useDeleteModal<PodKind>(pod);
  return <button onClick={launchDeleteModal}>{t('Delete Pod')}</button>
}
```

参数名称	描述
resource	要删除的资源。
redirectTo	（可选）在删除资源后重定向到的位置。
message	（可选）要在界面中显示的消息。
btnText	（可选）在删除按钮显示的文本。
deleteAllResources	（可选）删除同一类型的所有资源的功能。

返回

启动一个模态来删除资源的功能。

useLabelsModel
提供回调的 hook，以启动模态来编辑 Kubernetes 资源标签。

Example

```
const PodLabelsButton = ({ pod }) => {
```

```
  const { t } = useTranslation();
  const launchLabelsModal = useLabelsModal<PodKind>(pod);
  return <button onClick={launchLabelsModal}>{t('Edit Pod Labels')}</button>
}
```

参数名称	描述
resource	用于编辑标签的资源，这是 K8sResourceCommon 类型的对象。

返回

启动一个模态来编辑资源标签的功能。

useActiveNamespace
提供当前活跃的命名空间和回调来设置活跃命名空间的 hook。

Example

```
const Component: React.FC = (props) => {
  const [activeNamespace, setActiveNamespace] = useActiveNamespace();
  return <select
    value={activeNamespace}
    onChange={(e) => setActiveNamespace(e.target.value)}
  >
    {
      // ...namespace options
    }
  </select>
}
```

返回

包含当前活跃命名空间和 setter 回调的元组。

PerspectiveContext
弃用： 使用提供的 **usePerspectiveContext** 替代。创建视角上下文。

参数名称	描述
PerspectiveContextType	带有活跃视角和 setter 的对象

useAccessReviewAllowed
弃用：使用来自 **@console/dynamic-plugin-sdk** 的 **useAccessReview** 替代。hook，提供有关用户对给定资源的访问权限的允许状态。它返回 **isAllowed** 布尔值。

参数名称	描述
resourceAttributes	访问查看的资源属性

参数名称	描述
impersonate	模拟详情

useSafetyFirst
弃用：此 hook 与控制台功能无关。在可以卸载给定组件时，确保安全 asynchronnous 设置 React 状态的 hook。它返回一个数组，它带有一对状态值及其 set 功能。

参数名称	描述
initialState	初始状态值

YAMLEditor
已弃用：一个基本 lazy 加载的 YAML 编辑器，带有悬停的帮助和完成。

Example

```
<React.Suspense fallback={<LoadingBox />}>
  <YAMLEditor
    value={code}
  />
</React.Suspense>
```

参数名称	描述
value	代表要呈现的 yaml 代码的字符串。
options	Monaco 编辑器选项.
minHeight	有效 CSS 高度中的值的最小编辑器高度。
showShortcuts	布尔值，以显示编辑器顶部的快捷方式。
toolbarLinks	编辑器顶部的工具栏链接部分上 ReactNode rendered 的数组。
onChange	代码更改事件的回调。
onSave	触发命令 CTRL / CMD + S 时调用的回调。
ref	对 **{ editor?: IStandaloneCodeEditor }** 的响应参考。使用 **editor** 属性，您可以访问所有方法来控制编辑器。

7.5.3. 对动态插件进行故障排除

如果您在加载插件时遇到问题，请参阅此故障排除提示列表。

- 运行以下命令，在控制台 Operator 配置中启用了插件，并且您的插件名称是输出：

```
$ oc get console.operator.openshift.io cluster -o jsonpath='{.spec.plugins}'
```

  - 在 **Administrator 视角**中的 **Overview** 页面的状态卡中验证已启用的插件。如果插件最近启用，您必须刷新浏览器。

- 通过以下方法验证您的插件服务是否健康：
  - 验证您的插件 pod 状态正在运行，容器已就绪。
  - 验证服务标签选择器与 pod 和目标端口匹配是否正确。
  - 在控制台 pod 或集群中的另一个 pod 终端中，从服务 curl **plugin-manifest.json**。
- 验证 **ConsolePlugin** 资源名称 (**consolePlugin.name**) 与 **package.json** 中使用的插件名称匹配。
- 在 **ConsolePlugin** 资源中验证您的服务名称、命名空间、端口和路径是否已正确声明。
- 验证您的插件服务使用 HTTPS 和服务证书。
- 验证控制台 pod 日志中是否有证书或连接错误。
- 验证插件依赖的功能标志没有被禁用。
- 验证您的插件没有在 **package.json** 中不满足 **consolePlugin.dependencies** 的问题。
  - 这包括控制台版本依赖项或依赖其他插件。在浏览器中为您的插件的名称过滤 JS 控制台，以查看日志记录的消息。
- 验证 nav 扩展视角或部分 ID 中没有拼写错误。
  - 可以加载您的插件，但如果 ID 不正确，则缺少 nav 项。编辑 URL 尝试直接导航到插件页面。
- 验证没有网络策略阻止从控制台 Pod 到插件服务的流量。
  - 如有必要，调整网络策略以允许 openshift-console 命名空间中的控制台 pod 向服务发出请求。
- 在开发人员工具浏览器的 **Console** 选项卡中，验证要在浏览器中载入的动态插件列表。
  - 评估 **window.SERVER_FLAGS.consolePlugins**，以查看 Console frontend 上的动态插件。

其他资源

- 了解服务用证书

第 8 章 WEB 终端

8.1. 安装 WEB 终端

您可以使用 OpenShift Container Platform OperatorHub 中列出的 Web Terminal Operator 来安装 Web 终端。安装 Web Terminal Operator 时，会自动安装命令行配置（如 **DevWorkspace** CRD）所需的自定义资源定义（CRD）。打开 web 终端时，web 控制台会创建所需的资源。

先决条件

- 已登陆到 OpenShift Container Platform Web 控制台。
- 有集群管理员权限。

流程

1. 在 Web 控制台的 **Administrator** 视角中，导航到 **Operators → OperatorHub**。
2. 使用 **Filter by keyword** 复选框在目录中搜索 Web Terminal Operator，然后点 **Web Terminal** 标题。
3. 参阅 **Web Terminal** 页面中有关 Operator 的简单描述，然后点击 **Install**。
4. 在 **Install Operator** 页面中，保留所有字段的默认值。
   - **Update Channel** 菜单中的 **fast** 选项启用 Web Terminal Operator 最新版本的安装。
   - **Installation Mode** 菜单中的 **All namespaces on the cluster** 选项可让 Operator 监视并可供集群中的所有命名空间使用。
   - **Installed Namespace** 菜单中的 **openshift-operators** 选项会在默认的 **openshift-operators** 命名空间中安装 Operator。
   - **Approval Strategy** 菜单中的 **Automatic** 选项确保以后对 Operator 的升级由 Operator Lifecycle Manager 自动处理。
5. 点 **Install**。
6. 在 **Installed Operators** 页面中，点 **View Operator** 来验证 **Installed Operators** 页面中是否列出了 Operator。

   注意

   Web Terminal Operator 将 DevWorkspace Operator 安装为依赖项。

7. 安装 Operator 后，刷新页面以查看控制台 masthead 中的命令行终端图标()。

8.2. 配置 WEB 终端

您可以为 web 终端配置超时和镜像设置，无论是当前会话，则为所有用户会话（如果您是集群管理员）。

8.2.1. 为会话配置 web 终端超时

您可以更改当前会话的 web 终端的默认超时时间。

先决条件

- 您可以访问安装了 Web Terminal Operator 的 OpenShift Container Platform 集群。
- 已登陆到 web 控制台。

流程

1. 点 Web 终端图标()。
2. 可选：为当前会话设置 web 终端超时：
   a. 点 Timeout。
   b. 在出现的字段中，输入超时值。
   c. 从下拉列表中选择超时时间为 **Seconds**,**Minutes**,**Hours**, 或 **Milliseconds**。
3. 可选：为要使用的 web 终端选择一个自定义镜像。
   a. 点 Image。
   b. 在出现的字段中，输入您要使用的镜像的 URL。
4. 点 **Start** 使用指定的超时设置启动一个终端实例。

8.2.2. 为所有用户配置 web 终端超时

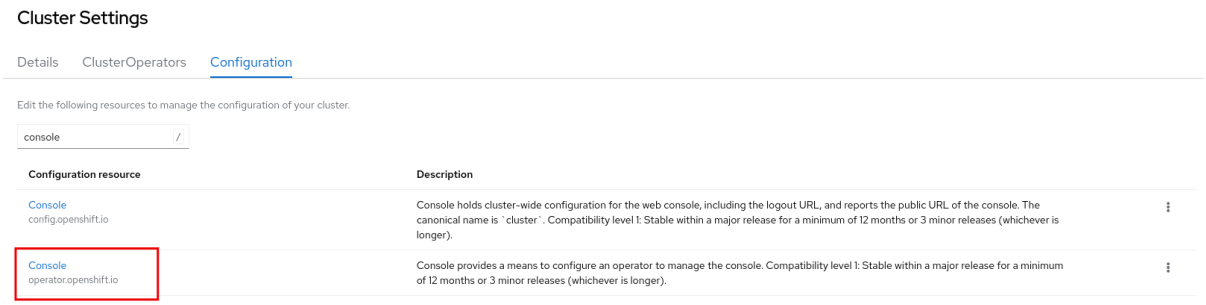
您可以使用 Web 控制台的 **Administrator** 视角为所有用户设置默认 web 终端超时时间。

先决条件

- 有集群管理员权限并登录到 web 控制台。
- 已安装 Web Terminal Operator。

流程

1. 在 **Administrator** 视角中，进入 **Administration → Cluster Settings**。
2. 在 **Cluster Settings** 页面中，点 **Configuration** 选项卡。
3. 在 **Configuration** 页面中，点带有描述 **operator.openshift.io** 的 **Console** 配置资源。



4. 从 **Action** 下拉列表中，选择 **Customize**，以打开 **Cluster 配置**页面。
5. 点 **Web Terminal** 选项卡，打开 **Web Terminal Configuration** 页面。
6. 设置一个超时值。从下拉列表中选择一个时间间隔，单位为 **Seconds**,**Minutes**,**Hours**, 或 **Milliseconds**。
7. 点击 **Save**。

8.2.3. 为会话配置 web 终端镜像

您可以为当前会话的 web 终端更改默认镜像。

先决条件

- 您可以访问安装了 Web Terminal Operator 的 OpenShift Container Platform 集群。
- 已登陆到 web 控制台。

流程

1. 点 Web 终端图标()。
2. 点 **Image** 以显示 web 终端镜像的高级配置选项。
3. 输入您要使用的镜像的 URL。
4. 点 **Start** 使用指定的镜像设置启动一个终端实例。

8.2.4. 为所有用户配置 web 终端镜像

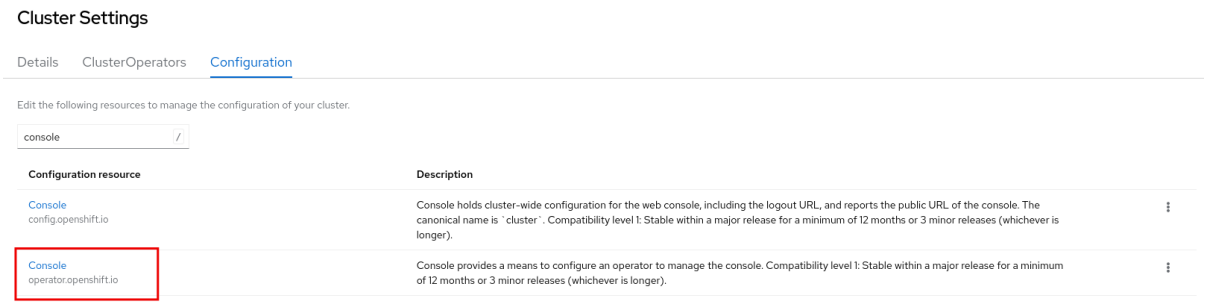
您可以使用 Web 控制台的 **Administrator** 视角为所有用户设置默认 web 终端镜像。

先决条件

- 有集群管理员权限并登录到 web 控制台。
- 已安装 Web Terminal Operator。

流程

1. 在 **Administrator** 视角中，进入 **Administration → Cluster Settings**。
2. 在 **Cluster Settings** 页面中，点 **Configuration** 选项卡。
3. 在 **Configuration** 页面中，点带有描述 **operator.openshift.io** 的 **Console** 配置资源。



4. 从 **Action** 下拉列表中，选择 **Customize**，以打开 **Cluster 配置**页面。
5. 点 **Web Terminal** 选项卡，打开 **Web Terminal Configuration** 页面。
6. 输入您要使用的镜像的 URL。
7. 点击 **Save**。

8.3. 使用 WEB 终端

您可以在 web 控制台中启动内嵌的命令行终端实例。此终端实例预安装了与集群交互的通用 CLI 工具，如 **oc**、**kubectl**、**odo**、**kn**、**tkn**、**helm** 和 **subctl**。它还包含正在处理的项目的上下文，并自动记录您使用凭证的项目。

8.3.1. 访问 Web 终端

安装 Web Terminal Operator 后，您可以访问 Web 终端。初始化 web 终端后，您可以在 web 终端中使用预安装的 CLI 工具，如 **oc**、**kubectl**、**odo**、**kn**、**tkn**、**helm** 和 **subctl**。您可以从在终端中运行的命令列表中选择这些命令，以重新运行这些命令。这些命令可在多个终端会话中保留。Web 终端保持打开，直到您关闭浏览器窗口或标签页。

先决条件

- 您可以访问 OpenShift Container Platform 集群，并登录到 web 控制台。
- 在集群中安装了 Web Terminal Operator。

流程

1. 要启动 web 终端，请在控制台的 masthead 中点命令行终端图标()。在 **Command line terminal** 窗格中会显示 web 终端实例。此实例使用您的凭证自动登录。
2. 如果在当前会话中没有选择项目，请从 **Project** 下拉列表中选择创建 **DevWorkspace** CR 的项目。默认情况下会选择当前项目。

注意

- 只有在不存在 **DevWorkspace** CR 时才会创建 DevWorkspace CR。
- **openshift-terminal** 项目是集群管理员使用的默认项目。它们没有选择其他项目的选项。Web Terminal Operator 将 DevWorkspace Operator 安装为依赖项。

3. 可选：为当前会话设置 web 终端超时：
   a. 点 Timeout。
   b. 在出现的字段中，输入超时值。
   c. 从下拉列表中选择超时时间为 **Seconds**,**Minutes**,**Hours**, 或 **Milliseconds**。
4. 可选：为要使用的 web 终端选择一个自定义镜像。
   a. 点 Image。
   b. 在出现的字段中，输入您要使用的镜像的 URL。
5. 点 **Start** 使用所选项目初始化 Web 终端。
6. 点 **+** 在控制台中 web 终端中打开多个标签页。

8.4. 对 WEB 终端进行故障排除

8.4.1. Web 终端和网络策略

如果集群配置了网络策略，web 终端可能无法启动。要初始化 Web 终端实例，Web Terminal Operator 必须与 Web 终端的 pod 通信以验证它是否正在运行，OpenShift Container Platform Web 控制台需要发送信息才能在终端中自动登录到集群。如果任一步骤失败，Web 终端将无法初始化，终端面板看似处于加载状态。

要避免这个问题，请确保用于终端的网络策略允许从 **openshift-console** 和 **openshift-operators** 命名空间进行入站数据。

8.5. 卸载 WEB 终端

卸载 Web Terminal Operator 不会删除安装 Operator 时创建的任何自定义资源定义 (CRD) 或受管资源。为了安全起见，您必须手动卸载这些组件。通过删除这些组件，您可以保存集群资源，因为在卸载 Operator 时终端不会闲置。

卸载 web 终端需要两步：

1. 卸载 Web Terminal Operator 和安装 Operator 时添加的相关自定义资源 (CR)。
2. 卸载 DevWorkspace Operator 及其作为 Web Terminal Operator 依赖项添加的相关自定义资源。

8.5.1. 删除 Web Terminal Operator

您可以通过删除 Web Terminal Operator 和 Operator 使用的自定义资源来卸载 web 终端。

先决条件

- 您可以使用集群管理员权限访问 OpenShift Container Platform 集群。
- 已安装 **oc** CLI。

流程

1. 在 web 控制台的 **Administrator** 视角中，导航到 **Operators → Installed Operators**。

2. 滚动过滤器列表或在 **Filter by name** 框中输入关键字以查找 Web Terminal Operator。

3. 点击 Web Terminal Operator 的 Options 菜单  ，然后选择 **Uninstall Operator**。

4. 在 **Uninstall Operator** 确认对话框中，点 **Uninstall** 从集群中删除 Operator、Operator 部署和 pod。Operator 会停止运行，并且不再接收更新。

8.5.2. 删除 DevWorkspace Operator

要完全卸载 web 终端，还必须删除 DevWorkspace Operator 和 Operator 使用的自定义资源。

重要

DevWorkspace Operator 是一个独立 Operator，可能需要作为集群中安装的其他 Operator 的依赖项。只有在确保不再需要 DevWorkspace Operator 时，才按照以下步骤操作。

先决条件

- 您可以使用集群管理员权限访问 OpenShift Container Platform 集群。
- 已安装 **oc** CLI。

流程

1. 删除 Operator 使用的 **DevWorkspace** 自定义资源，以及任何相关的 Kubernetes 对象：

```
$ oc delete devworkspaces.workspace.devfile.io --all-namespaces --all --wait
```

```
$ oc delete devworkspaceroutings.controller.devfile.io --all-namespaces --all --wait
```

警告

如果此步骤未完成，则终结器很难完全卸载 Operator。

2. 删除 Operator 使用的 CRD：

警告

DevWorkspace Operator 提供了使用转换 Webhook 的自定义资源定义 (CRD)。无法删除这些 CRD 可能会导致集群中的问题。

```
$ oc delete customresourcedefinitions.apiextensions.k8s.io
devworkspaceroutings.controller.devfile.io
```

```
$ oc delete customresourcedefinitions.apiextensions.k8s.io
devworkspaces.workspace.devfile.io
```

```
$ oc delete customresourcedefinitions.apiextensions.k8s.io
devworkspacetemplates.workspace.devfile.io
```

```
$ oc delete customresourcedefinitions.apiextensions.k8s.io
devworkspaceoperatorconfigs.controller.devfile.io
```

3. 验证所有涉及的自定义资源定义都已移除。以下命令不应该显示任何输出：

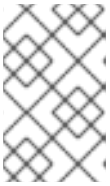
```
$ oc get customresourcedefinitions.apiextensions.k8s.io | grep "devfile.io"
```

4. 删除 **devworkspace-webhook-server** 部署、变异并验证 Webhook：

```
$ oc delete deployment/devworkspace-webhook-server -n openshift-operators
```

```
$ oc delete mutatingwebhookconfigurations controller.devfile.io
```

```
$ oc delete validatingwebhookconfigurations controller.devfile.io
```



注意

如果您在没有删除变异并验证 Webhook 的情况下删除 **devworkspace-webhook-server** 部署，则无法使用 **oc exec** 命令在集群中的容器中运行命令。删除 Webhook 后，您可以再次使用 **oc exec** 命令。

5. 删除任何剩余的服务、secret 和配置映射。取决于具体的安装，以下命令中包含的一些资源可能不存在。

```
$ oc delete all --selector app.kubernetes.io/part-of=devworkspace-
operator,app.kubernetes.io/name=devworkspace-webhook-server -n openshift-operators
```

```
$ oc delete serviceaccounts devworkspace-webhook-server -n openshift-operators
```

```
$ oc delete clusterrole devworkspace-webhook-server
```

```
$ oc delete clusterrolebinding devworkspace-webhook-server
```

6. 卸载 DevWorkspace Operator：

   a. 在 web 控制台的 **Administrator** 视角中，导航到 **Operators → Installed Operators**。

   b. 滚动过滤器列表或在 **Filter by name** 框中输入关键字以查找 DevWorkspace Operator。

c. 点 Operator 的 Options 菜单  ，然后选择 **Uninstall Operator**。

d. 在 **Uninstall Operator** 确认对话框中，点 **Uninstall** 从集群中删除 Operator、Operator 部署和 pod。Operator 会停止运行，并且不再接收更新。

第 9 章 在OPENSHIFT CONTAINER PLATFORM中禁用WEB控制台

您可以禁用OpenShift Container Platform Web控制台。

9.1. 先决条件

- 部署一个OpenShift Container Platform集群。

9.2. 禁用WEB控制台

您可以通过编辑**consoles.operator.openshift.io**资源来禁用Web控制台。

- 编辑 **consoles.operator.openshift.io** 资源：

```
$ oc edit consoles.operator.openshift.io cluster
```

以下示例显示了资源中可以修改的参数：

```
apiVersion: operator.openshift.io/v1
kind: Console
metadata:
  name: cluster
spec:
  managementState: Removed (1)
```

(1) 将**managementState**参数值设置为**Removed**以禁用Web控制台。此参数的其他有效值是**Managed**（启用由集群控制的控制台），**Unmanaged**（启用由用户控制管理的Web控制台）。

第 10 章 在 WEB 控制台中创建快速启动指南

如果您要为 OpenShift Container Platform Web 控制台创建快速启动指南，请按照以下步骤保留所有快速启动的用户体验。

10.1. 了解快速开始

快速开始是用户任务的指导教程。在 Web 控制台中，您可以在 **Help** 菜单下快速启动访问。它们在使用应用程序、Operator 或其他产品时特别有用。

快速开始主要由任务和步骤组成。每个任务都有多个步骤，每个快速开始都有多个任务。例如：

- 任务 1
  - 第 1 步
  - 第 2 步
  - 第 3 步
- 任务 2
  - 第 1 步
  - 第 2 步
  - 第 3 步
- 任务 3
  - 第 1 步
  - 第 2 步
  - 第 3 步

10.2. 快速启动用户工作流

当您与现有快速启动指南交互时，这是预期的工作流体验：

1. 在 **Administrator** 或 **Developer** 视角中，点击 **Help** 图标并选择 **Quick Starts**。
2. 点快速启动卡。
3. 在出现的面板中点 **Start**。
4. 完成屏幕的说明，然后点 **Next**。
5. 在出现 **Check your work** 模块时，回答问题以确认您成功完成了该任务。
   a. 如果您选择 **Yes**，点 **Next** 来继续到下一个任务。
   b. 如果您选择 **No**，重复任务说明并再次检查您的工作。
6. 重复以上第 1 到 6 步，以便快速完成剩余的任务。
7. 完成最后的任务后，点 **Close** 关闭快速启动。

10.3. 快速启动组件

快速开始由以下部分组成：

- **Card**：提供快速启动基本信息的 catalog 标题，其中包括标题、描述、时间提交和完成状态
- **Introduction**：概述快速开始的目标和任务
- **Task headings**：快速启动中每个任务的超链接标题
- **Check your work module**：一个模块用户使用一个模块来确认他们成功完成了任务，然后到快速启动的下一个任务为止
- **Hints**：帮助用户识别产品特定部分的动画
- **Buttons**
  - **Next and back buttons**：用来浏览快速开始任务里的步骤和模块的按钮
  - **Final screen buttons**：关闭快速启动，返回到快速启动中的先前任务，并查看所有快速启动

快速启动的主要内容包括以下部分：

- **Card copy**
- **简介**
- **Task steps**
- **Modals and in-app messaging**
- **Check your work module**

10.4. 快速开始

OpenShift Container Platform 引入了由 **ConsoleQuickStart** 对象定义的快速启动自定义资源。operator 和管理员可以使用此资源来快速使用集群。

先决条件

- 您必须具有集群管理员特权。

流程

1. 要创建新快速启动，请运行：

```
$ oc get -o yaml consolequickstart spring-with-s2i > my-quick-start.yaml
```

2. 运行：

```
$ oc create -f my-quick-start.yaml
```

3. 根据本文档中介绍的指南更新 YAML 文件。
4. 保存您的编辑。

10.4.1. 查看快速启动 API 文档

流程

- 要查看快速启动 API 文档，请运行：

```
$ oc explain consolequickstarts
```

运行 **oc explain -h** 以了解有关 **oc explain** 使用的更多信息。

10.4.2. 将快速启动 CR 中的元素映射到快速启动 CR

本节帮助您视觉地映射快速启动自定义资源（CR）的部分，使其出现在 web 控制台中快速启动的自定义资源（CR）中。

10.4.2.1. conclusion 元素

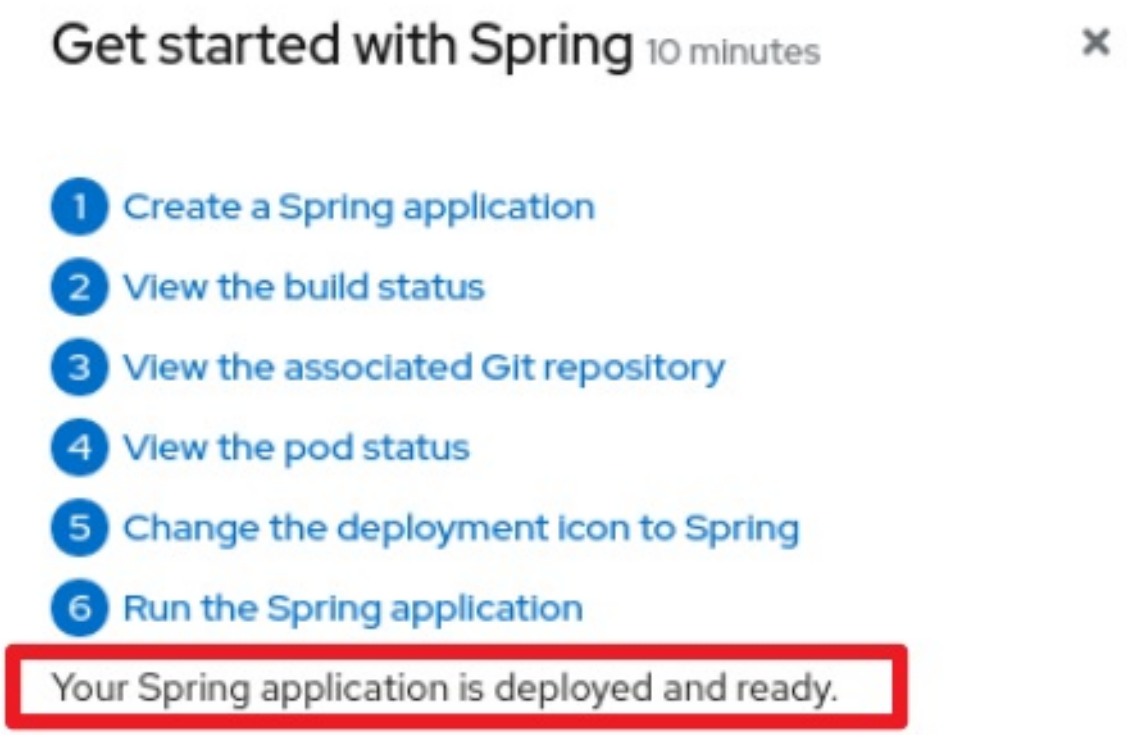
查看 YAML 文件中的 conclusion 元素

```
...
summary:
  failed: Try the steps again.
  success: Your Spring application is running.
title: Run the Spring application
conclusion: >-
  Your Spring application is deployed and ready. 1
```

1 conclusion 文本

在 web 控制台中查看 conclusion 元素

最后会出现在快速开始的最后部分。



10.4.2.2. description 元素

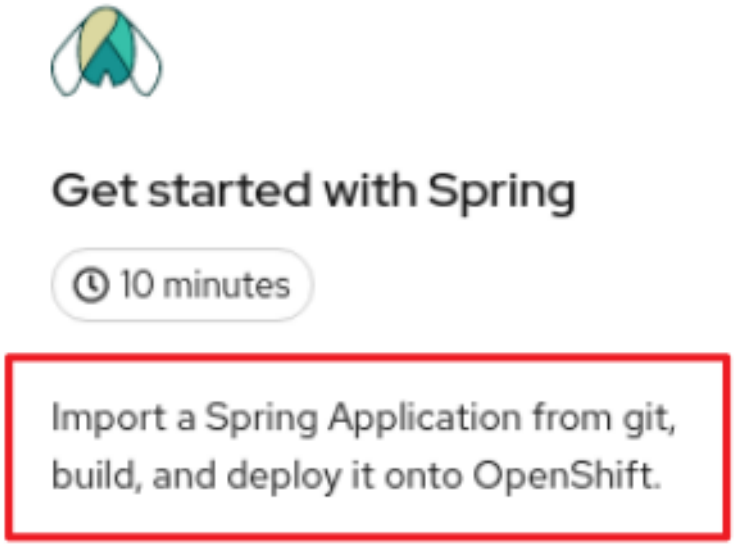
查看 YAML 文件中的 description 元素

```
apiVersion: console.openshift.io/v1
kind: ConsoleQuickStart
metadata:
  name: spring-with-s2i
spec:
  description: 'Import a Spring Application from git, build, and deploy it onto OpenShift.' (1)
...
```

(1) description 文本

在 web 控制台中查看 description 元素

这个描述会出现在**快速开始**页的介绍中。



10.4.2.3. displayName 元素

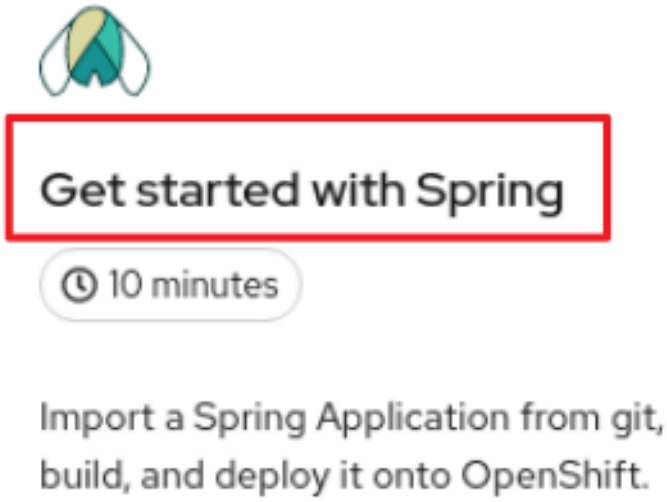
查看 YAML 文件中的 displayName 元素

```
apiVersion: console.openshift.io/v1
kind: ConsoleQuickStart
metadata:
  name: spring-with-s2i
spec:
  description: 'Import a Spring Application from git, build, and deploy it onto OpenShift.'
  displayName: Get started with Spring 1
  durationMinutes: 10
```

1 **displayName** 文本。

在 web 控制台中查看 displayName 元素

显示名称会出现在**快速**启动页的介绍中。



10.4.2.4. durationMinutes 元素

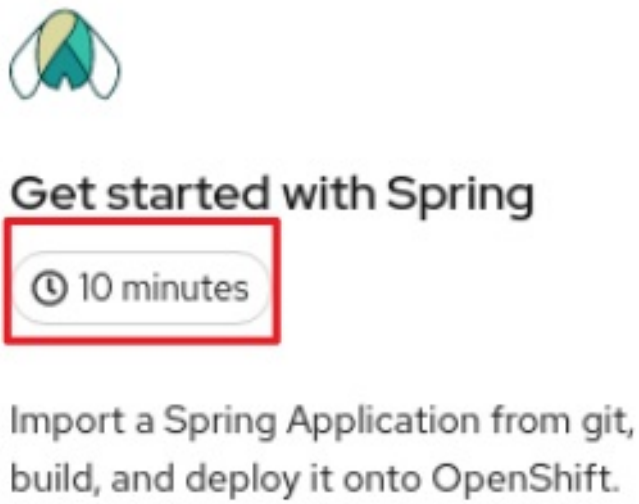
在 YAML 文件中查看 durationMinutes 元素

```
apiVersion: console.openshift.io/v1
kind: ConsoleQuickStart
metadata:
  name: spring-with-s2i
spec:
  description: 'Import a Spring Application from git, build, and deploy it onto OpenShift.'
  displayName: Get started with Spring
  durationMinutes: 10 (1)
```

(1) **durationMinutes** 值，以分钟为单位。这个值定义了快速启动完成所需的时间。

在 web 控制台中查看 durationMinutes 元素

durationMinutes 元素会出现在**快速开始**页的介绍中。



10.4.2.5. icon 元素

查看 YAML 文件中的 icon 元素

```
...
spec:
  description: 'Import a Spring Application from git, build, and deploy it onto OpenShift.'
  displayName: Get started with Spring
  durationMinutes: 10
  icon: >-   1
```

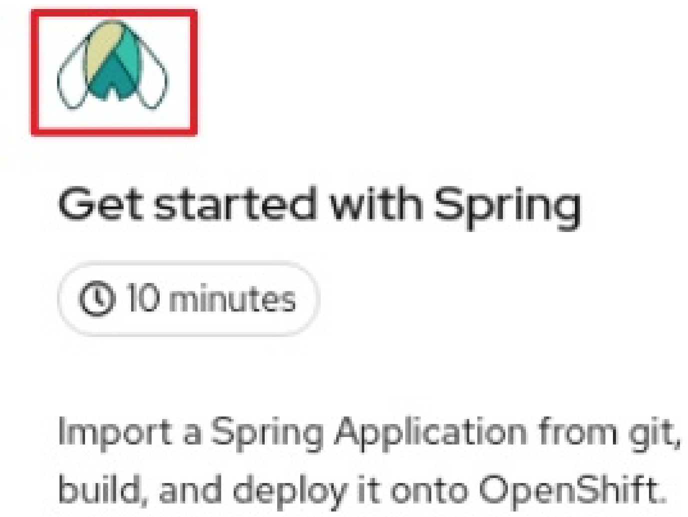

JMYXllcl8xIiBkYXRhLW5hbWU9IkxheWVyIDEiIHZpZXdCb3g9IjAgMCAxMDI0IDEwMjQiPjxkZWZzPjxzd
HlsZT4uY2xzLTF7ZmlsbDojMTUzZDNjO30uY2xzLTJ7ZmlsbDojZDhkYTlkO30uY2xzLTN7ZmlsbDojNT
hjMGE4O30uY2xzLTR7ZmlsbDojZmZmO30uY2xzLTV7ZmlsbDojM2Q5MTkxO308L3N0eWxlPjwvZGV
mcz48dGl0bGU+c25vd2Ryb3BfaWNvbl9yZ2JfZGVmYXVsdDwvdGl0bGU+PHBhdGggY2xhc3M9ImNsc
y0xIiBkPSJNMTAxMi42OSw1OTNjLTExLjEyLTM4LjA3LTMxLTczLTU5LjIxLTEwMy44LTkuNS0xMS4zL
TIzLjIxLTI4LjI5LTM5LjA2LTQ3Ljk0QzgzMy41MywzNDEsNzQ1LjM3LDIzNC4xOCw2NzQsMTY4Ljk0Yy
01LTUuMjYtMTAuMjYtMTAuMzEtMTUuNjUtMTUuMDdhMjQ2LjQ5LDI0Ni40OSwwLDAsMC0zNi41NS
0yNi44LDE4Mi41LDE4Mi41LDAsMCwwLTIwLjMtMTEuNzcsMjAxLjUzLDIwMS41MywwLDAsMC00My4
xOS0xNUExNTUuMjQsMTU1LjI0LDAsMCwwLDUyOCw5NS4yYy02Ljc2LS42OC0xMS43NC0uODEtM
TQuMzktLjgxaDBsLTEuNjIsMC0xLjYyLDBhMTc3LjMsMTc3LjMsMCwwLDAtMzEuNzcsMy4zNSwyMDg
uMjMsMjA4LjIzLDAsMCwwLTU2LjEyLDE3LjU2LDE4MSwxODEsMCwwLDAtMjAuMjcsMTEuNzUsMjQ
3LjQzLDI0Ny40MywwLDAsMC0zNi41NywyNi44MUMzNjAuMjUsMTU4LjYyLDM1NSwxNjMuNjgsMzUw
LDE2OWMtNzEuMzUsNjUuMjUtMTU5LjUsMTcyLTI0MC4zOSwyNzIuMjhDOTMuNzMsNDYwLjg4LDg
wLDQ3Ny44Nyw3MC41Miw0ODkuMTcsNDI0LDUyLjQzLDU1NC45LDExLjMxLDU5MywuNz
IsNjI5LjIyLTEuNzMsNjY3LjY5LDQsNzA3LjMxLDE1LDc4Mi40OSw1NS43OCw4NTkuMTIsMTE4LjkzLDcxLD
kyMy4wOWEyMiwyMiwwLDAsMCwxNS41OSw2LjUyaDEuODNsMS44MywwYTIyLDIyLDAsMCwwLDE1LjU4LTYuNTJjNjMuMTYtNjMuMTEsMTM4LjI2LDE2Mi
EsMTEwLTc5LjU3LDE0My40OC0xNTUuNiwzLjkzLjkxLTguODgsNy45NS0xOC4wNSwxMi4yLTI3LjQzLjQzLjk1LjU2LjkwLjkwLi4xMS0uMDguMzcuMjgu
uNDIsOC41NCwxMS4zOSwxNi4yM2MzMS44NSw0MC45MSw3NS4xMiw2NC42NywwMzIuMzIsNzIuNj
NsMTguOCwyLjYyLDQuOTUtMTguMzNjNjMMuMjYtNDkuMDcsMzUuMy05MC44NSw1MC42NC0xMT
YuMTksMTUuMzQsMjUuMzQsMzcuMzgsNjcuMTEsNTAuNjQsMTE2LjE5bDUuMTgsMTguMzMsMTguOC0yL
jYyYzU3LjItOCwxMDAuNDctMzEuNzIsMTMyLjMyLTcyLjYzYTYtNy42OCwxMS4zOS0xNi4yM2M0LjI1LjI1LTIy
DkuMzgsOC4yOSxxOC41NSxMi4yLDI3LjQzLDMzLjQ5LDc2LDYyLjQyLDE0MS42OSxNDMuNDgs
MTU1LjZsMS44MS4zMWgxLjg5YTIlLDIyLDAsMCwxLDE1LjU5LTYuNTJjNjMuMTUtNjQsMTAzLjk1LT
E0MC42LDExNC44OS0yMTUuNzhDMTAyNS43MDAyNS43Myw2NjcuNjksMTAyMy4yOCw2MjkuMjIsMTAxMi42O
Sw1OTNaIi8+PHBhdGggY2xhc3M9ImNscy0yIiBkPSJNMzY0LjE1LDE4NS4yM2MxNy44Ny0xNi40LDM
0LjctMzAuMTUsNDkuNzctNDAuMTFhMjEyLDIxLDIwLDAsMSw2NS45My0yNS43M0ExOTgsMTk4LDA
sMCwxLDUxMiwxMTYuMjdhMTk2LjExLDE5Ni4xMSwwLDAsMSwzMiwzLjFjNC41LjkxLDkuMzYsMi4wN
wxNC41MywzLjUyLDYwLjQxLDIwLjQ4LDg0LjkyLDkxLjA1LTQ3LjQ0LDI0OC4wNi0yOC43NSwzNC4x
Mi0xNDAuNywxOTQuODQtMTg0LjY2LDI2OC40MmE2MzAuODYsNjMwLjg2LDAsMCwwLTMzLjIyLD
U4LjMyQzI3Niw2NTUuMzQsMjY1LjQsNTk4LDI2NS40LDUyMC4yOSwyNjUuNCwzNDAuNjEsMzExLjY5
5LDI0MC43NCwzNjQuMTUsMTg1LjIzWiIvPjxwYXRoIGNsYXNzPSJjbHMtMyIgZD0iTTUyNy41NCwzO
DQuODNjODQuMDYtOTkuNywxMTYuMDYtMTc3LjI4LDk1LjIyLTIzMC43NCwxMS42Miw4LjY5LDI0LD
E5LjIsMzcuMDYsMzEuMTMsNTIuNDgsNTUuNSw5OC43OCwxNTUuMzgsOTguNzgsMzM1LjA3LDAs
NzcuNzEtMTAuNiwxMzUuMDUtMjcuNzcsMTc3LjRhNjI4LjczLDYyOC43MywwLDAsMC0zMy4yMy0
4zMmMtMzktNjUuMjYtMTMxLjQ1LTE5OS0xNzEuOTMtMjUyLjI3QzUyNi4zMywzODYuMjksNTI3LjM
NS41Miw1MjcuNTQsMzg0LjgzWiIvPjxwYXRoIGNsYXNzPSJjbHMtNCIgZD0iTTEzNC41OCw5MDguM
DdoLS4wNmEuMzkuMzksMCwwLDEtLjI3LS4xMWMtMTE5LjUyLTEyMS4wNy0xNTUtMjg3LjQtNDcuN
TQtNDA0LjU4LDM0LjYzLTQ1LDEyMC0xNTEuNiwyMDIuNzUtNzUuMjlsLjI1LjE5LTMuMTMsNy02LjEyLDE
0LjI1LTguOTIsMjEuNjktMjQuMzQsNjUuMTAtMTE5LjE3LjU2LDUzLDEzOC4yLTE1NiwxLDE4MDgzLDE
zLDU1NTgsMTA2LDE2LjU5LDE0Ny4xNEEzMDcuNDksMzA3LjQ5LDAsMCwwLDI4MC45MSw3MjQ
MjM3LDgxNi44NzEgMDIyLjUNzMMTAsNDAuNDQ5LDQlMwMTAwMMTgyNzgsMzAwLjQ
scy01IiBkPSJNNTgzLjQzLDgxMy43OUM1NjAuMTgsNzI3LjcyLDUxMiw2NjQuMTUsNTEyLDY2NC4xN
XMtNDguMTcsNjMuNTctNzEuNDMsMTQ5LjY0Yy00OC40NS02Ljc0LTEwMC45MS0yNy41Mi0xMzUuNy
NjYtOTEuNTMtMTAxLjMsMC45LS1.MjA1LS0xMS4zMi0xNjkuNzI4LjMxOC0xMDgyMC4zNC4yNS4zNy1
NjYtOTEuMThhNjQ1LDY0NS4yLDI0MC42NS42OCwwLDAsMSwyOS4xNS0yNS42NS0xMzUuNS4yMy4yM2M

```
zOC02My42MywxMjYuNC0xOTEuMzcsMTY3LjEyLTI0NS42Niw0MC43MSw1NC4yOCwxMjkuMSwxO
DIsMTY3LjEyLDI0NS42NmwuMTkuMzMuMjEuMzJhNjQ1LjY4LDY0NS42OCwwLDAsMSwzOS41Nyw
3MS41NEM2ODQuMzQsNzg2LjI3LDYzMS44OCw4MDcuMDUsNTgzLjQzLDgxMy43OVoiLz48cGF0a
CBjbGFzcz0iY2xzLTQiIGQ9Ik04ODkuNzUsOTA4YS4zOS4zOSwwLDAsMS0uMjcuMTFoLS4wNkM4M
DcuMDcsODkzLjkzLDc4Nyw4MTYuODgsNzQzLjA5LDcyM2EzMDcuNDksMzA3LjQ5LDAsMCwwLDIwL
jQ1LTU1LjU0YzExLTQxLjExLDE2LjU5LTkwLjYxLDE2LjU5LTE0Ny4xNCwwLTkzLjA4LTEyLjMzLTE3M
y0zNi42Ni0yMzcuNHEtNC4yMi0xMS4xNi04LjkzLTIxLjdjODIuNzUsOTAuNTksMTY4LjEyLDIwMS4wNS
wyMDIuNzUsMjQyLjE5QzEwNDQuNzksNjIwLjU2LDEwMDkuMjcsNzg2Ljg5LDg4OS43NSw5MDhaIi8+
PC9zdmc+Cg==
...
```


1 定义为 base64 值的图标。

在 web 控制台中查看 icon 元素

这个描述会出现在**快速开始**页的介绍中。



10.4.2.6. introduction 元素

查看 YAML 文件中的 introduction 元素

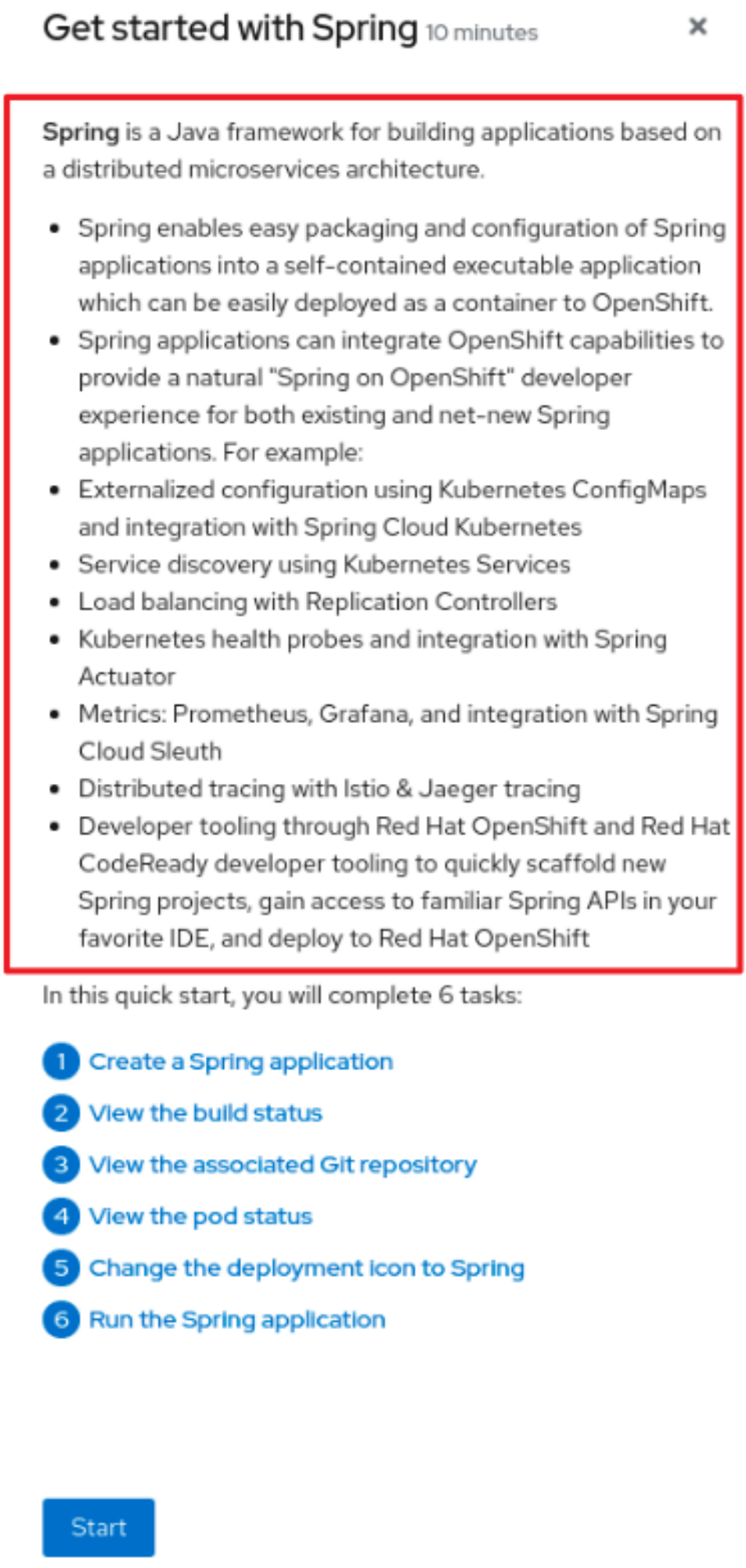
```
...
 introduction: >- (1)
   **Spring** is a Java framework for building applications based on a distributed microservices
architecture.

   - Spring enables easy packaging and configuration of Spring applications into a self-contained
executable application which can be easily deployed as a container to OpenShift.

   - Spring applications can integrate OpenShift capabilities to provide a natural "Spring on
OpenShift" developer experience for both existing and net-new Spring applications. For example:

   - Externalized configuration using Kubernetes ConfigMaps and integration with Spring Cloud
Kubernetes

   - Service discovery using Kubernetes Services
```

```
    - Load balancing with Replication Controllers

    - Kubernetes health probes and integration with Spring Actuator

    - Metrics: Prometheus, Grafana, and integration with Spring Cloud Sleuth

    - Distributed tracing with Istio & Jaeger tracing

    - Developer tooling through Red Hat OpenShift and Red Hat CodeReady developer tooling to
quickly scaffold new Spring projects, gain access to familiar Spring APIs in your favorite IDE, and
deploy to Red Hat OpenShift
...
```

1 简介介绍了快速启动并列出其中的任务。

在 web 控制台中查看 introduction 元素

点一个快速启动卡后，一个侧边面板滑盘将快速启动并列出它里面的任务。


Get started with Spring 10 minutes
Spring is a Java framework for building applications based on a distributed microservices architecture.
Spring enables easy packaging and configuration of Spring applications into a self-contained executable application which can be easily deployed as a container to OpenShift.
Spring applications can integrate OpenShift capabilities to provide a natural "Spring on OpenShift" developer experience for both existing and net-new Spring applications. For example:
Externalized configuration using Kubernetes ConfigMaps and integration with Spring Cloud Kubernetes
Service discovery using Kubernetes Services
Load balancing with Replication Controllers
Kubernetes health probes and integration with Spring Actuator
Metrics: Prometheus, Grafana, and integration with Spring Cloud Sleuth
Distributed tracing with Istio & Jaeger tracing
Developer tooling through Red Hat OpenShift and Red Hat CodeReady developer tooling to quickly scaffold new Spring projects, gain access to familiar Spring APIs in your favorite IDE, and deploy to Red Hat OpenShift
In this quick start, you will complete 6 tasks:
1 Create a Spring application
2 View the build status
3 View the associated Git repository
4 View the pod status
5 Change the deployment icon to Spring
6 Run the Spring application
Start

10.4.3. 为快速启动添加自定义图标

为所有快速启动提供了默认图标。您可以提供自己的自定义图标。

流程

1. 查找您要用作自定义图标的 **.svg** 文件。
2. 使用在线工具将文本转换为 base64。
3. 在 YAML 文件中，添加 **icon: >-**，然后在下一行中包含 **data:image/svg+xml;base64**，后面接的是 base64 转换的输出。例如：

```
icon: >-


ciIHJvbGU9ImltZyIgdmlld.
```

10.4.4. 限制对快速开始的访问

并不是所有的快速开始都应该对所有人可用。YAML 文件的 **accessReviewResources** 部分提供限制对快速启动的访问的能力。

只有有权创建 **HelmChartRepository** 资源的用户，才能访问快速开始，使用以下配置：

```
accessReviewResources:
  - group: helm.openshift.io
    resource: helmchartrepositories
    verb: create
```

只有用户具有列出 Operator 组和软件包清单（因此能够安装 Operator）时允许用户访问快速开始，使用以下配置：

```
accessReviewResources:
  - group: operators.coreos.com
    resource: operatorgroups
    verb: list
  - group: packages.operators.coreos.com
    resource: packagemanifests
    verb: list
```

10.4.5. 连接到其他快速开始

流程

- 在 YAML 文件的 **nextQuickStart** 部分，提供您要链接的快速开始的 **name** 而不是 **displayName**。例如：

```
nextQuickStart:
  - add-healthchecks
```

10.4.6. 支持的标签快速启动

使用这些标签在标记中写入快速开始内容。标记将转换为 HTML。

Tag	描述
'b',	粗体文本。
'img',	嵌入图像。
'i',	斜体文本。
'strike',	带有横线贯穿的文本。
's',	较小的文本
'del',	较小的文本。
'em',	加重文本。
'strong',	重要文本。
'a',	anchor 标签。
'p',	段落文本。
'h1',	1 级标题。
'h2',	2 级标题。
'h3',	3 级标题。
'h4',	4 级标题。
'ul',	一个没有顺序的列表。
'ol',	一个有顺序的列表。
'li',	一个列表项。
'code',	代码文本。
'pre',	预格式化的文本块。
'button',	文本中的一个按钮。

10.4.7. 快速入门突出显示 markdown 参考

高亮显示（或提示）功能可让快速入门包含可突出显示并模拟 web 控制台组件的链接。

markdown 语法包含：

- 括起的链接文本
- **highlight** 关键字，后跟您要动画的元素的 ID

10.4.7.1. 视角切换器

```
[Perspective switcher]{{highlight qs-perspective-switcher}}
```

10.4.7.2. Administrator 视角导航链接

```
[Home]{{highlight qs-nav-home}}
[Operators]{{highlight qs-nav-operators}}
[Workloads]{{highlight qs-nav-workloads}}
[Serverless]{{highlight qs-nav-serverless}}
[Networking]{{highlight qs-nav-networking}}
[Storage]{{highlight qs-nav-storage}}
[Service catalog]{{highlight qs-nav-servicecatalog}}
[Compute]{{highlight qs-nav-compute}}
[User management]{{highlight qs-nav-usermanagement}}
[Administration]{{highlight qs-nav-administration}}
```

10.4.7.3. Developer 视角导航链接

```
[Add]{{highlight qs-nav-add}}
[Topology]{{highlight qs-nav-topology}}
[Search]{{highlight qs-nav-search}}
[Project]{{highlight qs-nav-project}}
[Helm]{{highlight qs-nav-helm}}
```

10.4.7.4. 常用导航链接

```
[Builds]{{highlight qs-nav-builds}}
[Pipelines]{{highlight qs-nav-pipelines}}
[Monitoring]{{highlight qs-nav-monitoring}}
```

10.4.7.5. Masthea 链接

```
[CloudShell]{{highlight qs-masthead-cloudshell}}
[Utility Menu]{{highlight qs-masthead-utilitymenu}}
[User Menu]{{highlight qs-masthead-usermenu}}
[Applications]{{highlight qs-masthead-applications}}
[Import]{{highlight qs-masthead-import}}
[Help]{{highlight qs-masthead-help}}
[Notifications]{{highlight qs-masthead-notifications}}
```

10.4.8. 代码片段 markdown 参考

当 web 控制台的快速入门中包括了一个 CLI 代码片段时，您可以它。要使用这个功能，您必须首先安装 Web Terminal Operator。如果您没有安装 Web Terminal Operator，则 web 终端中执行的 web 终端和代

码片段操作将不存在。另外，无论您是否安装了 Web Terminal Operator，您都可以将代码片段复制到剪贴板中。

10.4.8.1. 内联代码片段的语法

```
`code block`{{copy}}
`code block`{{execute}}
```

注意

如果使用 **execute** 语法，无论您是否安装了 Web Terminal Operator，都会出现 **Copy to clipboard** 操作。

10.4.8.2. 多行代码片段的语法

```
```
multi line code block
```{{copy}}

```
multi line code block
```{{execute}}
```

10.5. 快速开始内容指南

10.5.1. Card copy

您可以在快速开始卡上自定义标题及描述，但您无法自定义状态。

- 将您的描述长度限制为一到两句。
- 从操作动词开始，并告知用户其目的。正确的示例：

  > Create a serverless application.

10.5.2. 简介

点一个快速启动卡后，一个侧边面板滑盘将快速启动并列出它里面的任务。

- 使您的简介内容更加简洁、明确、易于理解。
- 说明快速开始的目的。用户应在开始之前了解快速开始的目的。
- 为用户提供一个操作，而不是快速开始。
  - **正确的示例**：

    > In this quick start, you will deploy a sample application to {product-title}.

  - **不正确的示例**：

    > This quick start shows you how to deploy a sample application to {product-title}.

–

- 根据特性的复杂程度，简介应保持在最多四到五句。介绍不要太长。
- 列出简介内容后快速开始的任务，并以操作动词启动每个任务。不要指定任务数量，因为每次添加或删除任务时都需要更新副本。
  - **正确的示例**：

    > Tasks to complete: Create a serverless application; Connect an event source; Force a new revision

  - **不正确的示例**：

    > You will complete these 3 tasks: Creating a serverless application; Connecting an event source; Forcing a new revision

10.5.3. Task steps

用户点 **Start** 后会出现一系列步骤，它们必须执行这些操作来完成快速开始。

在编写任务步骤时遵循这些常规指南：

- 对于按钮和标签使用 "Click"。在选择框、单选按钮和下拉菜单中使用"选择"。
- 使用 "Click" 而不是 "Click on"
  - **正确的示例**：

    > Click OK.

  - **不正确的示例**：

    > Click on the OK button.

- 告诉用户如何在**管理**员和**开发者**视角间如何切换。即使您认为某个用户可能已经处于正确的视角，最好仍为用户提供相应的操作说明，使其确定位于正确的位置。
  示例：

  > Enter the Developer perspective: In the main navigation, click the dropdown menu and select Developer.
  > Enter the Administrator perspective: In the main navigation, click the dropdown menu and select Admin.

- 使用 "Location, action" 结构。告诉用户要做什么前先告诉用户到什么地方。
  - **正确的示例**：

    > In the node.js deployment, hover over the icon.

  - **不正确的示例**：

    > Hover over the icon in the node.js deployment.

- 保持您的产品术语大写一致。
- 如果您必须指定一个菜单类型或使用列表作为下拉菜单，使用 "dropdown"（一个单词，没有短横线）。
- 明确区分用户动作和产品功能的附加信息。
  - User action：

    > Change the time range of the dashboard by clicking the dropdown menu and selecting time range.

  - Additional information：

    > To look at data in a specific time frame, you can change the time range of the dashboard.

- 避免方向性的语言，如 "In the top-right corner, click the icon"。因为当 UI 布局改变时，方向语言就有可能变为不正确。另外，桌面用户对于具有不同屏幕大小的用户来说，方向可能是不正确的。反之，使用它的名称来标识项。
  - **正确的示例**：

    > In the navigation menu, click Settings.

  - **不正确的示例**：

    > In the left-hand menu, click Settings.

- 不要只使用颜色来标识项，如 "Click the gray circle"。颜色标识符对受限制的用户，尤其是无法识别颜色的用户可能不可用。相反，使用它的名称或复制来标识项目，如 button copy。
  - **正确的示例**：

    > The success message indicates a connection.

  - **不正确的示例**：

    > The message with a green icon indicates a connection.

- 一致性地使用第二人称（you）：
  - **正确的示例**：

    > Set up your environment.

  - **不正确的示例**：

    > Let's set up our environment.

10.5.4. Check your work module

- 用户完成一个步骤后会出现一个 **Check your work** 模块。这个模块提示用户回答"是"或对步骤结果没有问题，这使得他们有机会复核他们的工作。对于这个模块，您只需要写一个是或不需要问题。
  - 如果用户的回答是 **Yes**，会出现一个标记。
  - 如果用户的回答是 **No**， 会出现一个出错信息，其中包含相关文档的链接。然后，用户可以选择返回并再次进行尝试。

10.5.5. 格式化 UI 元素

使用以下指南格式化 UI 元素：

- 按钮、下拉菜单、标签、字段和其他 UI 控制的副本复制：在 UI 中写入副本并加粗体。
- 所有其他 UI 元素-包括页面、窗口和面板名称：在 UI 中写入该文件并加粗体。
- 代码或用户输入的文本：使用 monospaced 字体。
- 提示：如果包含到导航或 masthead 元素的提示，则使用您链接的文本。
- CLI 命令：使用 monospaced 字体。
- 在运行文本时，在命令中使用粗体 monospaced 字体。
- 如果参数或选项是一个变量值，使用 monospaced 字体。
- 参数使用粗体的 monospaced 字体，选项使用 monospaced 字体。

10.6. 其他资源

- 关于声音和音调的要求，请参考 PatternFly's brand voice and tone guidelines 。
- 关于其他 UX 内容指南，请参考 PatternFly 的 UX 编写风格指南 。

第 11 章 WEB 控制台中的可选功能和产品

您可以通过为现有工作流添加额外功能并通过产品集成来进一步自定义 OpenShift Container Platform Web 控制台。

11.1. 使用 OPERATOR 增强 OPENSHIFT CONTAINER PLATFORM WEB 控制台

集群管理员可以使用 OperatorHub 在 OpenShift Container Platform Web 控制台中在 OpenShift Container Platform Web 控制台中安装 Operator，为开发人员提供分层产品之外的自定义。例如，Web Terminal Operator 允许您在浏览器中打开使用常见 CLI 工具与集群交互的 Web 终端。

其他资源

- 了解 OperatorHub
- 安装 web 终端

11.2. WEB 控制台中的 RED HAT OPENSHIFT PIPELINES

Red Hat OpenShift Pipelines 是一个基于 Kubernetes 资源的云原生的持续集成和持续交付（continuous integration and continuous delivery，简称 CI/CD）的解决方案。使用 OpenShift Container Platform Web 控制台中的 OperatorHub 安装 Red Hat OpenShift Pipelines Operator。安装 Operator 后，您可以在 **Pipelines** 页面中创建并修改管道对象。

其他资源

- 在 web 控制台中使用 Red Hat OpenShift Pipelines
- Web 控制台中的管道执行统计

11.3. WEB 控制台中的 RED HAT OPENSHIFT SERVERLESS

Red Hat OpenShift Serverless 可让开发人员在 OpenShift Container Platform 上创建和部署无服务器、事件驱动的应用程序。您可以使用 OpenShift Container Platform Web 控制台 OperatorHub 安装 OpenShift Serverless Operator。

其他资源

- 通过 Web 控制台安装 OpenShift Serverless Operator

11.4. OPENSHIFT CONTAINER PLATFORM WEB 控制台中的 RED HAT DEVELOPER HUB

Red Hat Developer Hub 是一个可用于体验简化的开发环境的平台。Red Hat Developer Hub 由集中软件目录驱动，为您的微服务和基础架构提供效率。它使您的产品团队能够在没有任何影响的情况下提供质量代码。您可以快速开始以了解更多有关如何安装开发人员 hub 的信息。

11.4.1. 使用 OpenShift Container Platform Web 控制台安装 Red Hat Developer Hub

Web 控制台提供有关如何安装 Red Hat Developer Hub Operator 的说明的快速开始。

先决条件

- 您必须使用 **admin** 权限登录到 OpenShift Container Platform Web 控制台。

流程

1. 在 Administrator 视角的 **Overview** 页面中，点 **Getting started resources** 标题中的 **Install Red Hat Developer Hub (RHDH)**。
2. 此时会显示一个快速启动窗格，其中包含使用 Operator 安装 Red Hat Developer Hub 的说明。有关如何安装 Operator 的说明，创建一个 Red Hat Developer Hub 实例，并将您的实例添加到 **OpenShift Console Application** 菜单中。

验证

1. 您可以点显示的 **Application launcher** 链接来验证 **Application** 标签页是否可用。
2. 验证您的 Janus IDP 实例可以被打开。

其他资源

- Red Hat Developer Hub 产品文档