Red Hat Developer Hub 0.2

Topology plugin for Backstage

The Topology plugin for Backstage
The Topology plugin for Backstage
Abstract

The Topology plugin enables you to visualize the workloads such as Deployment, Job, Daemonset, Statefulset, CronJob, and Pods powering any service on the Kubernetes cluster.
Table of Contents

CHAPTER 1. TOPOLOGY PLUGIN FOR BACKSTAGE ........................................... 3
  1.1. FOR ADMINISTRATORS .......................................................... 3
      1.1.1. Installation ................................................................. 3
  1.2. FOR USERS ............................................................................ 9
      1.2.1. Using the Topology plugin in Backstage ......................... 9
CHAPTER 1. TOPOLOGY PLUGIN FOR BACKSTAGE

The Topology plugin enables you to visualize the workloads such as Deployment, Job, Daemonset, Statefulset, CronJob, and Pods powering any service on the Kubernetes cluster.

1.1. FOR ADMINISTRATORS

1.1.1. Installation

The Red Hat Plug-ins for Backstage (RHPIB) packages are hosted in a separate NPM registry, which is maintained by Red Hat. To use these packages, you must adjust your NPM configuration to pull the \texttt{@redhat} scoped packages:

\begin{verbatim}
  # update your .npmrc or .yarnrc file
  yarn config set "@redhat:registry" https://npm.registry.redhat.com
  # then pull a package
  yarn add @redhat/backstage-plugin-quay
\end{verbatim}

For more information, see \texttt{npm docs}.

Creating a \texttt{.npmrc} file ensures that all the packages are scoped under \texttt{@redhat} and are fetched from Red Hat's NPM registry, while the rest dependencies remain sourced from other registry.

Using this configuration, you can proceed with the installation of the individual packages.

Prerequisites

- The Kubernetes plugins including \texttt{@backstage/plugin-kubernetes} and \texttt{@backstage/plugin-kubernetes-backend} are installed and configured by following the \texttt{installation} and \texttt{configuration} guides.

- The Kubernetes plugin is configured and connects to the cluster using a \texttt{ServiceAccount}.

- The \texttt{ClusterRole} must be granted to \texttt{ServiceAccount} accessing the cluster. If you have the Backstage Kubernetes plugin configured, then the \texttt{ClusterRole} is already granted.

- The following must be added in \texttt{customResources} component in the \texttt{app-config.yaml} file to view the OpenShift route:

\begin{verbatim}
    kubernetes:
      ...
      customResources:
        - group: 'route.openshift.io'
          apiVersion: 'v1'
          plural: 'routes'
\end{verbatim}

Also, ensure that the route is granted a \texttt{ClusterRole}. You can use the following code to grant the \texttt{ClusterRole} to the route:

\begin{verbatim}
  ...
  apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRole
  metadata:
    name: backstage-read-only
\end{verbatim}
The following permission must be granted to the **ClusterRole** to be able to view the pod logs:

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

The following code must be added in **customResources** component in the `app-config.yaml` file to view the Tekton PipelineRuns list in the side panel and to view the latest PipelineRun status in the Topology node decorator:

```yaml
customResources:
  ...
  - group: 'tekton.dev'
    apiVersion: 'v1beta1'
    plural: 'pipelines'
  - group: 'tekton.dev'
    apiVersion: 'v1beta1'
    plural: 'pipelineruns'
  - group: 'tekton.dev'
    apiVersion: 'v1beta1'
    plural: 'taskruns'
```

Also, ensure that the Pipeline, PipelineRun, and TaskRun are granted a **ClusterRole**. You can use the following code to grant the **ClusterRole** to Pipeline, PipelineRun, and TaskRun:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: backstage-read-only
rules:
  ...
- apiGroups:
```
The following configuration must be added in `customResources` component in the `app-config.yaml` file to view the edit code decorator:

```yaml
- tekton.dev
resources:
- pipelines
- pipelineruns
- taskruns
verbs:
- get
- list
```

- The following annotations are added to workload resources in the `deployment.yaml` file to navigate to the GitHub repository of the associated application using the edit code decorator:

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

You can also add the following annotation to navigate to a specific branch:

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

If Red Hat OpenShift Dev Spaces (RHODS) is installed and configured and Git URL annotations are also added in the workload YAML file, then clicking on the edit code decorator redirects you to the RHODS instance. For more information about installing RHODS, see `Administration guide` of RHODS.
When you deploy your application using the OCP git import flow, then you do not need to add the labels as import flow to the workload YAML file. Otherwise, you would need to add the labels to the workload YAML file manually.

The labels are not similar to `backstage.io/edit-url` annotations as the added labels point to the source file of catalog entity metadata and is applied to Backstage catalog entity metadata YAML file, but not to the Kubernetes resources.

You can also add the `app.openshift.io/edit-url` annotation that you want to access using the decorator.

**TIP**

You can use the prepared manifest for a read-only ClusterRole, which provides access for both Kubernetes and Topology plugin.

- The following annotation is added to the entity’s `catalog-info.yaml` file to identify whether an entity contains the Kubernetes resources:

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

The following label is added to the resources so that the Kubernetes plugin receives the Kubernetes resources from the requested entity:

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

**NOTE**

When using the label selector, the mentioned labels must be present on the resource.

- The `backstage.io/kubernetes-namespace` annotation is added as follows to identify that the Kubernetes resources are using the defined namespace:

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

If the `backstage.io/kubernetes-namespace` annotation is added to the `catalog-info.yaml` file, then the RHODS instance is not accessible using the edit code decorator.

To retrieve the instance URL, CheCluster Custom Resource (CR) is required. The instance URL is not retrieved if the namespace annotation value is different from `openshift-devspaces` as CheCluster CR is created in `openshift-devspaces` namespace.

- A custom label selector is added, which Backstage uses to find the Kubernetes resources. The label selector takes precedence over the ID annotations. For example:

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

If you have multiple entities while RHODS is configured and want multiple entities to support the edit code decorator that redirects to the RHODS instance, you can add the
backstage.io/kubernetes-label-selector annotation to the catalog-info.yaml file for each entity as follows:

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

If you are using the previous custom label selector, then make sure that you add the following labels to your resources so that the Kubernetes plugin receives the Kubernetes resources from the requested entity:

```yaml
labels:
  component: che # add this label to your che cluster instance
```

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

You can also write your own custom query for the label selector with unique labels to differentiate your entities. However, you need to ensure that you add those labels to the resources associated with your entities including your CheCluster instance.

- The following label is added to workload resources in the deployment.yaml file to display runtime icon in the topology nodes:

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

Alternatively, you can include the following label to display the runtime icon:

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

The `<RUNTIME_NAME>` parameter in the previous example label supports the following values:

- django
- dotnet
- drupal
- go-gopher
- golang
- grails
- jboss
- jruby
- js
- nginx
- nodejs
- openjdk
- perl
- phalcon
- php
- python
- quarkus
- rails
- redis
- rh-spring-boot
- rust
- java
- rh-openjdk
- ruby
- spring
- spring-boot

Any other value for `<RUNTIME_NAME>` parameter results in icons not being rendered for the node.

- The following label is added to display the workload resources such as Deployments and Pods in a visual group:

```
yaml title="catalog-info.yaml"
labels:
  app.kubernetes.io/part-of: <GROUP_NAME>
```

- The following annotation is added to display the workload resources such as Deployments and Pods with a visual connector:

```
yaml title="catalog-info.yaml"
annotations:
  app.openshift.io/connects-to: '[["apiVersion": <RESOURCE_APIVERSION>,"kind": <RESOURCE_KIND>,"name": <RESOURCE_NAME>]]'
```

For more information about the labels and annotations, see Guidelines for labels and annotations for OpenShift applications.

Procedure
1. Install the Topology plugin using the following command:

   yarn workspace app add @redhat/backstage-plugin-topology

2. Enable TOPOLOGY tab in packages/app/src/components/catalog/EntityPage.tsx:

   ```tsx
   /* highlight-add-next-line */
   import { TopologyPage } from '@redhat/backstage-plugin-topology';
   
   const serviceEntityPage = (  
      <EntityPageLayout>  
        {/* ... */}  
        {/* highlight-add-start */}  
        <EntityLayout.Route path="/topology" title="Topology">  
          <TopologyPage />  
        </EntityLayout.Route>  
        {/* highlight-add-end */}  
      </EntityPageLayout>  
    );
   ```

1.2. FOR USERS

1.2.1. Using the Topology plugin in Backstage

Topology is a front-end plugin that enables you to view the workloads as nodes that power any service on the Kubernetes cluster.

Prerequisites

- Your Backstage application is installed and running.
- You have installed the Topology plugin. For the installation process, see Section 1.1.1, “Installation”.

Procedure

1. Open your Backstage application and select a component from the Catalog page.

2. Go to the TOPOLOGY tab and you can view the workloads such as Deployments, Pods as nodes.
3. Select a node and a pop-up appears on the right side, which contains two tabs: Details and Resources. The Details and Resources tab contain the associated information and resources of the node.

4. Click on the Open URL button on the top of a node.
When you click on the open URL button, it allows you to access the associated **Ingresses** and runs your application in a new tab.