Red Hat Developer Hub 0.2

Open Cluster Management plugin for Backstage

The Open Cluster Management plugin for Backstage

Last Updated: 2023-09-28

Red Hat Developer Hub 0.2 Open Cluster Management plugin for Backstage

The Open Cluster Management plugin for Backstage

Legal Notice

Abstract

The Open Cluster Management (OCM) plugin integrates your Backstage instance with the `MultiClusterHub` and `MultiCluster` engines of OCM.

Table of Contents

CHAPTER 1. OPEN CLUSTER MANAGEMENT PLUGIN FOR BACKSTAGE

The Open Cluster Management (OCM) plugin integrates your Backstage instance with the **MultiClusterHub** and **MultiCluster** engines of OCM.

1.1. CAPABILITIES

The OCM plugin has the following capabilities:

- All clusters represented as **ManagedCluster** in **MultiClusterHub** or MCE are discovered and imported into the Backstage catalog, such as:
  - Entity is defined as **kind: Resource** with **spec.type** set to **kubernetes-cluster**.
  - Links to the OpenShift Container Platform (OCP) console, OCM console, and OpenShift Cluster Manager are provided in **metadata.links**.
- Shows real-time data from OCM on the Resource entity page, including:
  - Cluster current status (up or down)
  - Cluster nodes status (up or down)
  - Cluster details (console link, OCP, and Kubernetes version)
  - Details about available compute resources on the cluster

1.2. FOR ADMINISTRATORS

1.2.1. 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 **@redhat** scoped packages:

```
# 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
```

For more information, see npm docs.

Creating a **.npmrc** file ensures that all the packages are scoped under **@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.

The OCM plugin is composed of two packages, including:

- **@redhat/backstage-plugin-ocm-backend** package connects the Backstage server to OCM. For setup process, see Section 1.2.1.2, "Setting up the OCM backend package" section.

- The **@redhat/backstage-plugin-ocm** package, which contains frontend components requires the ***-backend** package to be present and properly set up. For detailed instructions on setting up the backend, refer to the Section 1.2.1.3, "Setting up the OCM frontend package" section.

NOTE

If you are interested in Resource discovery and do not want any of the front-end components, then you can install and configure the **@redhat/backstage-plugin-ocm-backend** package only.

1.2.1.1. Prerequisites

- OCM is deployed and configured on a Kubernetes cluster.
- Kubernetes plugin for Backstage is installed.
- A **ClusterRole** is granted to **ServiceAccount** accessing the hub cluster as follows:

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: backstage-ocm-plugin
rules:
  - apiGroups:
      - cluster.open-cluster-management.io
    resources:
      - managedclusters
    verbs:
      - get
      - watch
      - list
  - apiGroups:
      - internal.open-cluster-management.io
    resources:
      - managedclusterinfos
    verbs:
      - get
      - watch
      - list
```

1.2.1.2. Setting up the OCM backend package

1. Install the OCM backend plugin using the following command:

```
yarn workspace backend add @redhat/backstage-plugin-ocm-backend
```

2. Configure the OCM backend plugin using one of the following configurations:

   - The OCM configuration provides the information about your hub. To use the OCM configuration, add the following code to your **app-config.yaml** file:

```
`yaml title="app-config.yaml"
    catalog:
      providers:
        ocm:
```

```
        env: # Key is reflected as provider ID. Defines and claims plugin instance ownership of entities
          name: # Name that the hub cluster will assume in Backstage Catalog (in OCM this is always local-cluster which can be confusing)
          url: # Url of the hub cluster API endpoint
          serviceAccountToken: # Token used for querying data from the hub
          skipTLSVerify: # Skip TLS certificate verification, defaults to false
          caData: # Base64-encoded CA bundle in PEM format
```

- If the Backstage Kubernetes plugin is installed and configured to connect to the hub cluster, then you can bind the both hub and Kubernetes configuration by providing the name of the hub in the **app-config.yaml** as follows:

```
```yaml title="app-config.yaml"
    kubernetes:
      serviceLocatorMethod:
        type: 'multiTenant'
      clusterLocatorMethods:
        - type: 'config'
          clusters:
            # highlight-next-line
            - name: <cluster-name>
            # ...

    catalog:
      providers:
        ocm:
          env: # Key is reflected as provider ID. Defines and claims plugin instance ownership of entities
            # highlight-next-line
            kubernetesPluginRef: <cluster-name> # Match the cluster name in kubernetes plugin config
```

Ensure that the Backstage uses a **ServiceAccount** token and the required permissions are granted as mentioned previously.

This is useful when you already use a Kubernetes plugin in your Backstage instance. Also, the hub cluster must be connected using the **ServiceAccount**.

For more information about the configuration, see Backstage Kubernetes plugin documentation.

3. Create a new plugin instance in **packages/backend/src/plugins/ocm.ts** file as follows:

```
```ts title="packages/backend/src/plugins/ocm.ts"
  import { Router } from 'express';

  import { createRouter } from '@redhat/backstage-plugin-ocm-backend';

  import { PluginEnvironment } from '../types';

  export default async function createPlugin(
    env: PluginEnvironment,
  ): Promise<Router> {
    return await createRouter({
      logger: env.logger,
```

```
    config: env.config,
  });
}
```

4. Import and plug the new instance into **packages/backend/src/index.ts** file:

```
```ts title="packages/backend/src/index.ts"
  /* highlight-add-next-line */
  import ocm from './plugins/ocm';

  async function main() {
    // ...
    const createEnv = makeCreateEnv(config);
    // ...
    /* highlight-add-next-line */
    const ocmEnv = useHotMemoize(module, () => createEnv('ocm'));
    // ...
    const apiRouter = Router();
    // ...
    /* highlight-add-next-line */
    apiRouter.use('/ocm', await ocm(ocmEnv));
    // ...
  }
```
```

5. Import the cluster **Resource** entity provider into the **catalog** plugin in the **packages/backend/src/plugins/catalog.ts** file. The scheduler also needs to be configured. Two configurations are possible here:

   a. Configure the scheduler inside the **app-config.yaml**:

```
```yaml title="app-config.yaml"
    catalog:
      providers:
        ocm:
          env:
            # ...
            # highlight-add-start
            schedule: # optional; same options as in TaskScheduleDefinition
              # supports cron, ISO duration, "human duration" as used in code
              frequency: { minutes: 1 }
              # supports ISO duration, "human duration" as used in code
              timeout: { minutes: 1 }
            # highlight-add-end
+
```

   and then use the configured scheduler

```
```ts title="packages/backend/src/index.ts"
    /* highlight-add-next-line */
    import { ManagedClusterProvider } from '@redhat/backstage-plugin-ocm-backend';

    export default async function createPlugin(
      env: PluginEnvironment,
    ): Promise<Router> {
```

```
  const builder = await CatalogBuilder.create(env);
  // ...
  /* highlight-add-start */
  const ocm = ManagedClusterProvider.fromConfig(env.config, {
    logger: env.logger,
    scheduler: env.scheduler,
  });
  builder.addEntityProvider(ocm);
  /* highlight-add-start */
  // ...
}
```

b. Add a schedule directly inside the **packages/backend/src/plugins/catalog.ts** file

```
```ts title="packages/backend/src/index.ts"
  /* highlight-add-next-line */
  import { ManagedClusterProvider } from '@redhat/backstage-plugin-ocm-backend';

  export default async function createPlugin(
    env: PluginEnvironment,
  ): Promise<Router> {
    const builder = await CatalogBuilder.create(env);
    // ...
    /* highlight-add-start */
    const ocm = ManagedClusterProvider.fromConfig(env.config, {
      logger: env.logger,
      schedule: env.scheduler.createScheduledTaskRunner({
        frequency: { minutes: 1 },
        timeout: { minutes: 1 },
      }),
    });
    builder.addEntityProvider(ocm);
    /* highlight-add-start */
    // ...
  }
```

6. Optional: Configure the default owner for the cluster entities in the catalog for a specific environment. For example, use the following code to set **foo** as the owner for clusters from **env** in the **app-config.yaml** catalog section:

```
`yaml title="app-config.yaml"
  catalog:
    providers:
      ocm:
        env:
          # highlight-next-line
          owner: user:foo
```

For more information about the default owner configuration, see upstream string references documentation.

1.2.1.3. Setting up the OCM frontend package

1. Install the OCM frontend plugin using the following command:
```

```
yarn workspace app add @redhat/backstage-plugin-ocm
```

2. Select the components that you want to use, such as:

- **OcmPage**: This is a standalone page or dashboard displaying all clusters as tiles. You can add **OcmPage** to **packages/app/src/App.tsx** file as follows:

```tsx title="packages/app/src/App.tsx"
/* highlight-add-next-line */
import { OcmPage } from '@redhat/backstage-plugin-ocm';

const routes = (
  <FlatRoutes>
    {/* ... */}
    {/* highlight-add-next-line */}
    <Route path="/ocm" element={<OcmPage logo={<Logo />} />} />
  </FlatRoutes>
);
```

You can also update navigation in **packages/app/src/components/Root/Root.tsx** as follows:

```tsx title="packages/app/src/components/Root/Root.tsx"
/* highlight-add-next-line */
import StorageIcon from '@material-ui/icons/Storage';

export const Root = ({ children }: PropsWithChildren<{}>) => (
  <SidebarPage>
    <Sidebar>
      <SidebarGroup label="Menu" icon={<MenuIcon />}>
        {/* ... */}
        {/* highlight-add-next-line */}
        <SidebarItem icon={StorageIcon} to="ocm" text="Clusters" />
      </SidebarGroup>
      {/* ... */}
    </Sidebar>
    {children}
  </SidebarPage>
);
```

- **ClusterContextProvider**: This component is a React context provided for OCM data, which is related to the current entity. The **ClusterContextProvider** component is used to display any data on the React components mentioned in **packages/app/src/components/catalog/EntityPage.tsx**:

```tsx title="packages/app/src/components/catalog/EntityPage.tsx"
/* highlight-add-start */
import {
  ClusterAvailableResourceCard,
  ClusterContextProvider,
  ClusterInfoCard,
} from '@redhat/backstage-plugin-ocm';

/* highlight-add-end */
```

```
const isType = (types: string | string[]) => (entity: Entity) => {
  if (!entity?.spec?.type) {
    return false;
  }
  return typeof types === 'string'
    ? entity?.spec?.type === types
    : types.includes(entity.spec.type as string);
};

export const resourcePage = (
  <EntityLayout>
    {/* ... */}
    {/* highlight-add-start */}
    <EntityLayout.Route path="/status" title="status">
      <EntitySwitch>
        <EntitySwitch.Case if={isType('kubernetes-cluster')}>
          <ClusterContextProvider>
            <Grid container direction="column" xs={6}>
              <Grid item>
                <ClusterInfoCard />
              </Grid>
              <Grid item>
                <ClusterAvailableResourceCard />
              </Grid>
            </Grid>
          </ClusterContextProvider>
        </EntitySwitch.Case>
      </EntitySwitch>
    </EntityLayout.Route>
    {/* highlight-add-end */}
  </EntityLayout>
);

export const entityPage = (
  <EntitySwitch>
    {/* ... */}
    {/* highlight-add-next-line */}
    <EntitySwitch.Case if={isKind('resource')} children={resourcePage} />
  </EntitySwitch>
);
```

In the previous codeblock, you can place the context provider into your **Resource** entity renderer, which is usually available in **packages/app/src/components/catalog/EntityPage.tsx** or in an imported component.

- **<ClusterInfoCard />**: This is an entity component displaying details of a cluster in a table:
- **<ClusterAvailableResourceCard />**: This is an entity component displaying the available resources on a cluster. For example, see **.status.capacity** of the **ManagedCluster** resource.

1.3. FOR USERS

1.3.1. Using the OCM plugin in Backstage

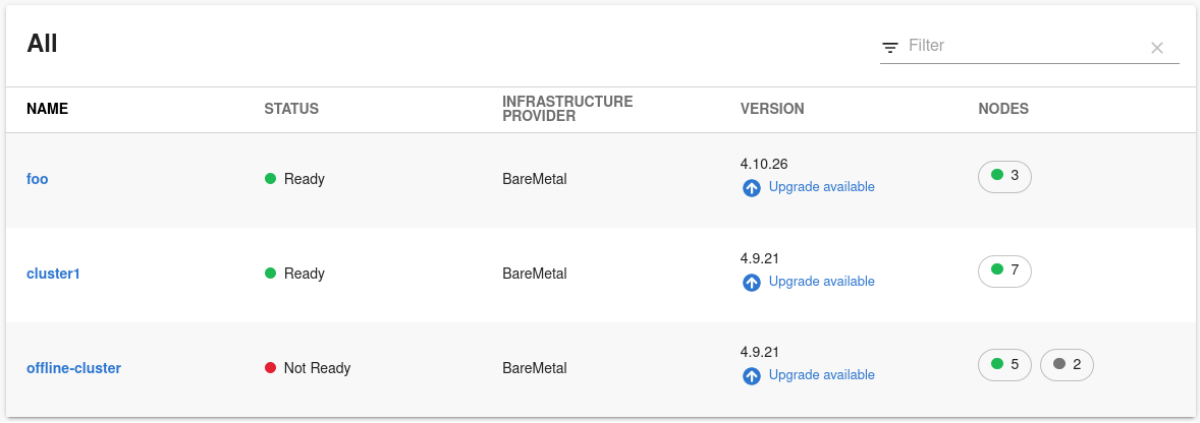
The OCM plugin integrates your Backstage instance with multi-cluster engines and displays real-time data from OCM.

1.3.1.1. Prerequisites

- Your Backstage application is installed and running.
- You have installed the OCM plugin. For the installation process, see Section 1.2.1, “Installation”.

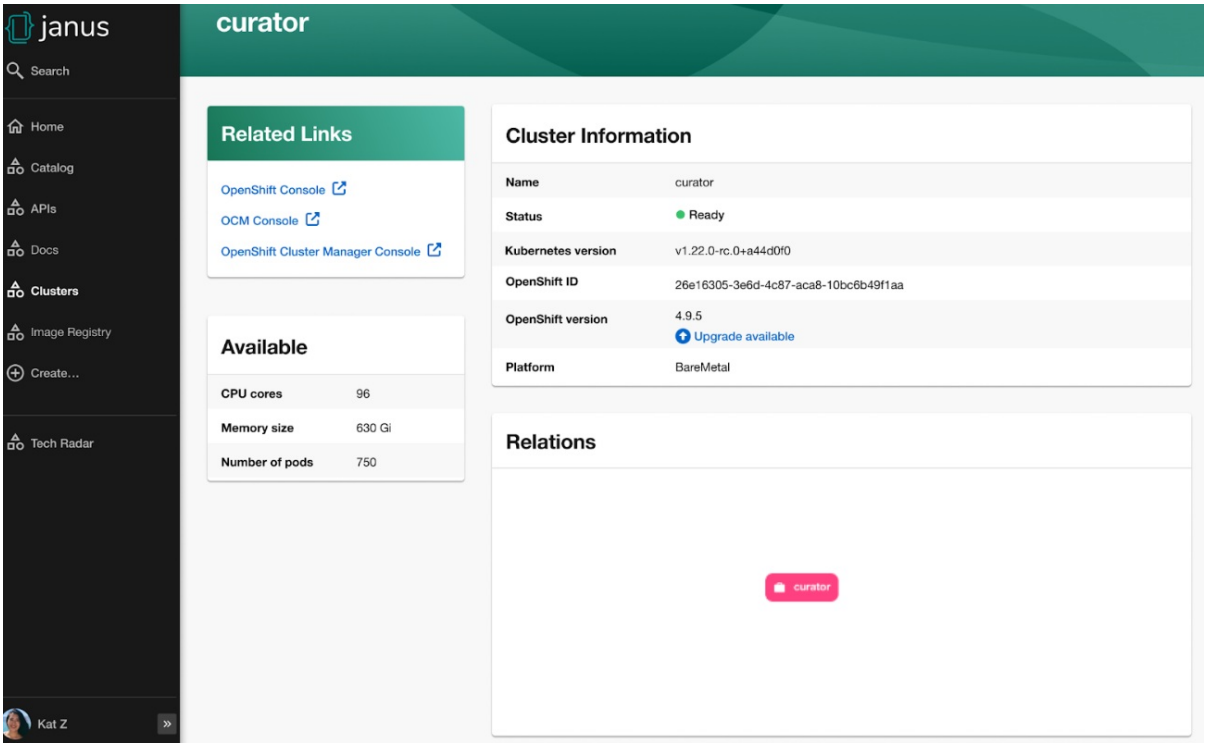
1.3.1.2. Procedure

1. Open your Backstage application.
2. Click the **Clusters** tab from the left-side panel to view the **Managed Clusters** page.
   The **Managed Clusters** page displays the list of clusters with additional information, such as status, infrastructure provider, associated OpenShift version, and available nodes.



   You can also upgrade the OpenShift version for a cluster using the **Upgrade available** option in the **VERSION** column.

3. Select a cluster from the **Managed Clusters** to view the related cluster information.
   You are redirected to the cluster-specific page, which consists of:
   - **Cluster Information**, such as name, status, accessed Kubernetes version, associated OpenShift ID and version, and accessed platform.
   - **Available** cluster capacity, including CPU cores, memory size, and number of pods.
   - **Related Links**, which enable you to access different consoles directly, such as OpenShift Console, OCM Console, and OpenShift Cluster Manager Console.
   - **Relations** card, which displays the visual representation of the cluster and associated dependencies.


janus
Search
Home
Catalog
APIs
Docs
Clusters
Image Registry
Create...
Tech Radar
Kat Z
curator
Related Links
OpenShift Console
OCM Console
OpenShift Cluster Manager Console
Available
CPU cores 96
Memory size 630 Gi
Number of pods 750
Cluster Information
Name curator
Status Ready
Kubernetes version v1.22.0-rc.0+a44d0f0
OpenShift ID 26e16305-3e6d-4c87-aca8-10bc6b49f1aa
OpenShift version 4.9.5
Upgrade available
Platform BareMetal
Relations
curator

CHAPTER 2. OPEN CLUSTER MANAGEMENT PLUGIN FOR BACKSTAGE

The Open Cluster Management (OCM) plugin integrates your Backstage instance with OCM.

For more information about OCM plugin, see the Open Cluster Management plugin documentation on GitHub.

CHAPTER 3. OPEN CLUSTER MANAGEMENT PLUGIN FOR BACKSTAGE

The Open Cluster Management (OCM) plugin integrates your Backstage instance with OCM.

For more information about OCM plugin, see the Open Cluster Management plugin documentation on GitHub.