Red Hat

Red Hat Developer Hub 1.6

Configuring dynamic plugins

Configuring dynamic plugins in Red Hat Developer Hub

Last Updated: 2025-09-25

Red Hat Developer Hub 1.6 Configuring dynamic plugins

Configuring dynamic plugins in Red Hat Developer Hub

Abstract

As a platform engineer, you can configure dynamic plugins in Red Hat Developer Hub (RHDH) to access your development infrastructure or software development tools.

Table of Contents

CHAPTER 1. INSTALLING ANSIBLE PLUG-INS FOR RED HAT DEVELOPER HUB

Ansible plug-ins for Red Hat Developer Hub deliver an Ansible-specific portal experience with curated learning paths, push-button content creation, integrated development tools, and other opinionated resources.

IMPORTANT

The Ansible plug-ins are a Technology Preview feature only.

Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend using them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information on Red Hat Technology Preview features, see Technology Preview Features Scope.

Additional detail on how Red Hat provides support for bundled community dynamic plugins is available on the Red Hat Developer Support Policy page.

To install and configure the Ansible plugins, see *Installing Ansible plug-ins for Red Hat Developer Hub* .

CHAPTER 2. ENABLING THE ARGO CD PLUGIN

You can use the Argo CD plugin to visualize the Continuous Delivery (CD) workflows in OpenShift GitOps. This plugin provides a visual overview of the status of the application, deployment details, commit message, author of the commit, container image promoted to environment and deployment history.

Prerequisites

- Add Argo CD instance information to your **app-config.yaml** configmap as shown in the following example:

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

  NOTE

  Avoid using a trailing slash in the **url**, as it might cause unexpected behavior.

- Add the following annotation to the entity **catalog-info.yaml** file to identify the Argo CD applications.

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

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

- (Optional) Add the following annotation to the entity's **catalog-info.yaml** file to switch between Argo CD instances as shown in the following example:

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

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

  NOTE

  If you do not set this annotation, the Argo CD plugin defaults to the first Argo CD instance configured in **app-config.yaml**.

Procedure

1. Add the following to your dynamic-plugins ConfigMap to enable the Argo CD plugin.

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

2.1. ENABLING ARGO CD ROLLOUTS

The optional Argo CD Rollouts feature enhances Kubernetes by providing advanced deployment strategies, such as blue-green and canary deployments, for your applications. When integrated into the backstage Kubernetes plugin, it allows developers and operations teams to visualize and manage Argo CD Rollouts seamlessly within the Backstage interface.

Prerequisites

- The Backstage Kubernetes plugin (**@backstage/plugin-kubernetes**) is installed and configured.
    - To install and configure Kubernetes plugin in Backstage, see Installaltion and Configuration guide.
- You have access to the Kubernetes cluster with the necessary permissions to create and manage custom resources and **ClusterRoles**.
- The Kubernetes cluster has the **argoproj.io** group resources (for example, Rollouts and AnalysisRuns) installed.

Procedure

1. In the **app-config.yaml** file in your Backstage instance, add the following **customResources** component under the **kubernetes** configuration to enable Argo Rollouts and AnalysisRuns:

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

2. Grant **ClusterRole** permissions for custom resources.

NOTE

- If the Backstage Kubernetes plugin is already configured, the **ClusterRole** permissions for Rollouts and AnalysisRuns might already be granted.
- Use the prepared manifest to provide read-only **ClusterRole** access to both the Kubernetes and ArgoCD plugins.

a. If the **ClusterRole** permission is not granted, use the following YAML manifest to create the **ClusterRole**:

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

a. Apply the manifest to the cluster using **kubectl**:

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

b. Ensure the **ServiceAccount** accessing the cluster has this **ClusterRole** assigned.

3. Add annotations to **catalog-info.yaml** to identify Kubernetes resources for Backstage.

a. For identifying resources by entity ID:

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

b. (Optional) For identifying resources by namespace:

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

c. For using custom label selectors, which override resource identification by entity ID or namespace:

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

NOTE

Ensure you specify the labels declared in **backstage.io/kubernetes-label-selector** on your Kubernetes resources. This annotation overrides entity-based or namespace-based identification annotations, such as **backstage.io/kubernetes-id** and **backstage.io/kubernetes-namespace**.

4. Add label to Kubernetes resources to enable Backstage to find the appropriate Kubernetes resources.

   a. Backstage Kubernetes plugin label: Add this label to map resources to specific Backstage entities.

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

   b. GitOps application mapping: Add this label to map Argo CD Rollouts to a specific GitOps application

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

NOTE

If using the label selector annotation (backstage.io/kubernetes-label-selector), ensure the specified labels are present on the resources. The label selector will override other annotations like kubernetes-id or kubernetes-namespace.

Verification

1. Push the updated configuration to your GitOps repository to trigger a rollout.
2. Open Red Hat Developer Hub interface and navigate to the entity you configured.
3. Select the **CD** tab and then select the **GitOps application**. The side panel opens.
4. In the **Resources** table of the side panel, verify that the following resources are displayed:
   - Rollouts
   - AnalysisRuns (optional)
5. Expand a rollout resource and review the following details:
   - The Revisions row displays traffic distribution details for different rollout versions.
   - The Analysis Runs row displays the status of analysis tasks that evaluate rollout success.

Additional resources

- The package path, scope, and name of the Red Hat ArgoCD plugin has changed since 1.2. For more information, see Breaking Changes in the *Red Hat Developer Hub release notes* .

- For more information on installing dynamic plugins, see Installing and viewing plugins in Red Hat Developer Hub.

CHAPTER 3. INSTALLING AND CONFIGURING THE JFROG ARTIFACTORY PLUGIN

JFrog Artifactory is a front-end plugin that displays the information about your container images stored in the JFrog Artifactory repository. The JFrog Artifactory plugin is preinstalled with Developer Hub and disabled by default. To use it, you need to enable and configure it first.

IMPORTANT

The JFrog Artifactory plugin is a Technology Preview feature only.

Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend using them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information on Red Hat Technology Preview features, see Technology Preview Features Scope.

Additional detail on how Red Hat provides support for bundled community dynamic plugins is available on the Red Hat Developer Support Policy page.

3.1. INSTALLATION

The JFrog Artifactory plugin is preinstalled in Developer Hub with basic configuration properties. To enable it, set the disabled property to **false** as follows:

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

3.2. CONFIGURATION

1. Set the proxy to the desired JFrog Artifactory server in the **app-config.yaml** file as follows:

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

2. Add the following annotation to the entity’s **catalog-info.yaml** file to enable the JFrog Artifactory plugin features in RHDH components:

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

CHAPTER 4. INSTALLING AND CONFIGURING KEYCLOAK

The Keycloak backend plugin, which integrates Keycloak into Developer Hub, has the following capabilities:

- Synchronization of Keycloak users in a realm.
- Synchronization of Keycloak groups and their users in a realm.

NOTE

The supported Red Hat Build of Keycloak (RHBK) version is **26.0**.

4.1. INSTALLATION

The Keycloak plugin is pre-loaded in Developer Hub with basic configuration properties. To enable it, set the **disabled** property to **false** as follows:

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

4.2. BASIC CONFIGURATION

To enable the Keycloak plugin, you must set the following environment variables:

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

4.3. ADVANCED CONFIGURATION

Schedule configuration

You can configure a schedule in the **app-config.yaml** file, as follows:

```
catalog:
  providers:
    keycloakOrg:
      default:
        # ...
        # highlight-add-start
```

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

NOTE

If you have made any changes to the schedule in the **app-config.yaml** file, then restart to apply the changes.

Keycloak query parameters

You can override the default Keycloak query parameters in the **app-config.yaml** file, as follows:

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

Communication between Developer Hub and Keycloak is enabled by using the Keycloak API. Username and password, or client credentials are supported authentication methods.

The following table describes the parameters that you can configure to enable the plugin under **catalog.providers.keycloakOrg.<ENVIRONMENT_NAME>** object in the **app-config.yaml** file:

Name	Description	Default Value	Required
baseUrl	Location of the Keycloak server, such as **https://localhost:8443/auth**.	""	Yes
realm	Realm to synchronize	**master**	No
loginRealm	Realm used to authenticate	**master**	No
username	Username to authenticate	""	Yes if using password based authentication
password	Password to authenticate	""	Yes if using password based authentication

Name	Description	Default Value	Required
clientId	Client ID to authenticate	""	Yes if using client credentials based authentication
clientSecret	Client Secret to authenticate	""	Yes if using client credentials based authentication
userQuerySize	Number of users to query at a time	**100**	No
groupQuerySize	Number of groups to query at a time	**100**	No

When using client credentials, the access type must be set to **confidential** and service accounts must be enabled. You must also add the following roles from the **realm-management** client role:

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

4.4. METRICS

The Keycloak backend plugin supports OpenTelemetry metrics that you can use to monitor fetch operations and diagnose potential issues.

4.4.1. Available Counters

Table 4.1. Keycloak metrics

Metric Name	Description
backend_keycloak_fetch_task_failure_count_total	Counts fetch task failures where no data was returned due to an error.
backend_keycloak_fetch_data_batch_failure_count_total	Counts partial data batch failures. Even if some batches fail, the plugin continues fetching others.

4.4.2. Labels

All counters include the **taskInstanceId** label, which uniquely identifies each scheduled fetch task. You can use this label to trace failures back to individual task executions.

Users can enter queries in the Prometheus UI or Grafana to explore and manipulate metric data.

In the following examples, a Prometheus Query Language (PromQL) expression returns the number of backend failures.

Example to get the number of backend failures associated with a **taskInstanceId**

```
backend_keycloak_fetch_data_batch_failure_count_total{taskInstanceId="df040f82-2e80-44bd-83b0-
06a984ca05ba"} 1
```

Example to get the number of backend failures during the last hour

```
sum(backend_keycloak_fetch_data_batch_failure_count_total) -
sum(backend_keycloak_fetch_data_batch_failure_count_total offset 1h)
```

NOTE

PromQL supports arithmetic operations, comparison operators, logical/set operations, aggregation, and various functions. Users can combine these features to analyze time-series data effectively.

Additionally, the results can be visualized using Grafana.

4.4.3. Exporting Metrics

You can export metrics using any OpenTelemetry-compatible backend, such as **Prometheus**.

See the Backstage OpenTelemetry setup guide for integration instructions.

4.5. LIMITATIONS

If you have self-signed or corporate certificate issues, you can set the following environment variable before starting Developer Hub:

NODE_TLS_REJECT_UNAUTHORIZED=0

NOTE

The solution of setting the environment variable is not recommended.

CHAPTER 5. INSTALLING AND CONFIGURING THE NEXUS REPOSITORY MANAGER PLUGIN

The Nexus Repository Manager plugin displays the information about your build artifacts in your Developer Hub application. The build artifacts are available in the Nexus Repository Manager.

IMPORTANT

The Nexus Repository Manager plugin is a Technology Preview feature only.

Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend using them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information on Red Hat Technology Preview features, see Technology Preview Features Scope.

Additional detail on how Red Hat provides support for bundled community dynamic plugins is available on the Red Hat Developer Support Policy page.

5.1. INSTALLATION

The Nexus Repository Manager plugin is pre-loaded in Developer Hub with basic configuration properties. To enable it, set the disabled property to **false** as follows:

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

5.2. CONFIGURATION

1. Set the proxy to the desired Nexus Repository Manager server in the **app-config.yaml** file as follows:

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

2. Optional: Change the base URL of Nexus Repository Manager proxy as follows:

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

3. Optional: Enable the following experimental annotations:

```
nexusRepositoryManager:
  experimentalAnnotations: true
```

4. Annotate your entity using the following annotations:

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

CHAPTER 6. INSTALLING AND CONFIGURING THE TEKTON PLUGIN

You can use the Tekton plugin to visualize the results of CI/CD pipeline runs on your Kubernetes or OpenShift clusters. The plugin allows users to visually see high level status of all associated tasks in the pipeline for their applications.

6.1. INSTALLATION

Prerequisites

- You have installed and configured the **@backstage/plugin-kubernetes** and **@backstage/plugin-kubernetes-backend** dynamic plugins.
- You have configured the Kubernetes plugin to connect to the cluster using a **ServiceAccount**.
- The **ClusterRole** must be granted for custom resources (PipelineRuns and TaskRuns) to the **ServiceAccount** accessing the cluster.

  NOTE

  If you have the RHDH Kubernetes plugin configured, then the **ClusterRole** is already granted.

- To view the pod logs, you have granted permissions for **pods/log**.
- You can use the following code to grant the **ClusterRole** for custom resources and pod logs:

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



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

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

- Add the following annotation to the entity’s **catalog-info.yaml** file to identify whether an entity contains the Kubernetes resources:

```
annotations:
  ...

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

- You can also add the **backstage.io/kubernetes-namespace** annotation to identify the Kubernetes resources using the defined namespace.

```
annotations:
  ...

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

- Add the following annotation to the **catalog-info.yaml** file of the entity to enable the Tekton related features in RHDH. The value of the annotation identifies the name of the RHDH entity:

```
annotations:
  ...

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

- Add a custom label selector, which RHDH uses to find the Kubernetes resources. The label selector takes precedence over the ID annotations.

```
annotations:
  ...

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

- Add the following label to the resources so that the Kubernetes plugin gets the Kubernetes resources from the requested entity:

```
labels:
  ...

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

NOTE

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

Procedure

- The Tekton plugin is pre-loaded in RHDH with basic configuration properties. To enable it, set the disabled property to false as follows:

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

CHAPTER 7. INSTALLING THE TOPOLOGY PLUGIN

7.1. INSTALLATION

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

Prerequisites

- You have installed and configured the @backstage/plugin-kubernetes-backend dynamic plugins.
- You have configured the Kubernetes plugin to connect to the cluster using a ServiceAccount.
- The **ClusterRole** must be granted to ServiceAccount accessing the cluster.

NOTE

If you have the Developer Hub Kubernetes plugin configured, then the **ClusterRole** is already granted.

Procedure

- The Topology plugin is pre-loaded in Developer Hub with basic configuration properties. To enable it, set the disabled property to false as follows:

  app-config.yaml fragment

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

7.2. CONFIGURING THE TOPOLOGY PLUGIN

7.2.1. Viewing OpenShift routes

Procedure

1. To view OpenShift routes, grant read access to the routes resource in the Cluster Role:

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

```
      - route.openshift.io
    resources:
      - routes
    verbs:
      - get
      - list
```

2. Also add the following in **kubernetes.customResources** property in your **app-config.yaml** file:

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

7.2.2. Viewing pod logs

Procedure

- To view pod logs, you must grant the following permission to the **ClusterRole**:

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

7.2.3. Viewing Tekton PipelineRuns

Procedure

1. To view the Tekton PipelineRuns, grant read access to the **pipelines**, **pipelinesruns**, and **taskruns** resources in the **ClusterRole**:

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

2. To view the Tekton PipelineRuns list in the side panel and the latest PipelineRuns status in the Topology node decorator, add the following code to the **kubernetes.customResources** property in your **app-config.yaml** file:

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

7.2.4. Viewing virtual machines

Prerequisites

1. The OpenShift Virtualization operator is installed and configured on a Kubernetes cluster. .Procedure

2. Grant read access to the **VirtualMachines** resource in the **ClusterRole**:

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

3. To view the virtual machine nodes on the topology plugin, add the following code to the **kubernetes.customResources** property in the **app-config.yaml** file:

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

7.2.5. Enabling the source code editor

To enable the source code editor, you must grant read access to the CheClusters resource in the **ClusterRole** as shown in the following example code:

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

To use the source code editor, you must add the following configuration to the **kubernetes.customResources** property in your **app-config.yaml** file:

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

7.3. MANAGING LABELS AND ANNOTATIONS FOR TOPOLOGY PLUGINS

7.3.1. Linking to the source code editor or the source

Add the following annotations to workload resources, such as Deployments to navigate to the Git repository of the associated application using the source code editor:

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

Add the following annotation to navigate to a specific branch:

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

NOTE

If Red Hat OpenShift Dev Spaces is installed and configured and Git URL annotations are also added to the workload YAML file, then clicking on the edit code decorator redirects you to the Red Hat OpenShift Dev Spaces instance.

NOTE

When you deploy your application using the OCP Git import flows, then you do not need to add the labels as import flows do that. Otherwise, you need to add the labels manually to the workload YAML file.

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

7.3.2. Entity annotation/label

For RHDH to detect that an entity has Kubernetes components, add the following annotation to the **catalog-info.yaml** file of the entity:

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

Add the following label to the resources so that the Kubernetes plugin gets the Kubernetes resources from the requested entity:

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

NOTE

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

7.3.3. Namespace annotation

Procedure

- To identify the Kubernetes resources using the defined namespace, add the **backstage.io/kubernetes-namespace** annotation:

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

  The Red Hat OpenShift Dev Spaces instance is not accessible using the source code editor if the **backstage.io/kubernetes-namespace** annotation is added to the **catalog-info.yaml** file.

To retrieve the instance URL, you require the CheCluster custom resource (CR). As the CheCluster CR is created in the openshift-devspaces namespace, the instance URL is not retrieved if the namespace annotation value is not openshift-devspaces.

7.3.4. Label selector query annotation

You can write your own custom label, which RHDH uses to find the Kubernetes resources. The label selector takes precedence over the ID annotations:

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

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

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

If you are using the previous label selector, you must add the following labels to your resources so that the Kubernetes plugin gets the Kubernetes resources from the requested entity:

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

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.

7.3.5. Icon displayed in the node

To display a runtime icon in the topology nodes, add the following label to workload resources, such as Deployments:

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

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

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

Supported values of **<RUNTIME_NAME>** include:

- 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

NOTE

Other values result in icons not being rendered for the node.

7.3.6. App grouping

To display workload resources such as deployments or pods in a visual group, add the following label:

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

7.3.7. Node connector

Procedure

To display the workload resources such as deployments or pods with a visual connector, add the following annotation:

+

```
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*.

CHAPTER 8. BULK IMPORTING GITHUB REPOSITORIES

IMPORTANT

These features are for Technology Preview only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend using them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information on Red Hat Technology Preview features, see Technology Preview Features Scope.

Red Hat Developer Hub can automate GitHub repositories onboarding and track their import status.

8.1. ENABLING AND GIVING ACCESS TO THE BULK IMPORT FEATURE

You can enable the Bulk Import feature for users and give them the necessary permissions to access it.

Prerequisites

- You have configured GitHub integration.

Procedure

1. The Bulk Import plugins are installed but disabled by default. To enable the **./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import-backend-dynamic** and **./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import** plugins, edit your **dynamic-plugins.yaml** with the following content:

   dynamic-plugins.yaml fragment

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

   See Installing and viewing plugins in Red Hat Developer Hub .

2. Configure the required **bulk.import** RBAC permission for the users who are not administrators as follows:

   rbac-policy.csv fragment

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

   Note that only Developer Hub administrators or users with the **bulk.import** permission can use the Bulk Import feature. See Permission policies in Red Hat Developer Hub .

Verification

- The sidebar displays a **Bulk Import** option.
- The **Bulk Import** page shows a list of **Added Repositories**.

8.2. IMPORTING MULTIPLE GITHUB REPOSITORIES

In Red Hat Developer Hub, you can select your GitHub repositories and automate their onboarding to the Developer Hub catalog.

Prerequisites

- You have enabled the Bulk Import feature and gave access to it .

Procedure

1. Click **Bulk Import** in the left sidebar.
2. Click the **Add** button in the top-right corner to see the list of all repositories accessible from the configured GitHub integrations.
    a. From the **Repositories** view, you can select any repository, or search for any accessible repositories. For each repository selected, a **catalog-info.yaml** is generated.
    b. From the **Organizations** view, you can select any organization by clicking **Select** in the third column. This option allows you to select one or more repositories from the selected organization.
3. Click **Preview file** to view or edit the details of the pull request for each repository.
    a. Review the pull request description and the **catalog-info.yaml** file content.
    b. Optional: when the repository has a **.github/CODEOWNERS** file, you can select the **Use CODEOWNERS file as Entity Owner** checkbox to use it, rather than having the **content-info.yaml** contain a specific entity owner.
    c. Click **Save**.
4. Click **Create pull requests**. At this point, a set of dry-run checks runs against the selected repositories to ensure they meet the requirements for import, such as:
    a. Verifying that there is no entity in the Developer Hub catalog with the name specified in the repository **catalog-info.yaml**
    b. Verifying that the repository is not empty
    c. Verifying that the repository contains a **.github/CODEOWNERS** file if the **Use CODEOWNERS file as Entity Owner** checkbox is selected for that repository
        - If any errors occur, the pull requests are not created, and you see a *Failed to create PR* error message detailing the issues. To view more details about the reasons, click **Edit**.
        - If there are no errors, the pull requests are created, and you are redirected to the list of added repositories.
5. Review and merge each pull request that creates a **catalog-info.yml** file.

Verification

- The **Added repositories** list displays the repositories you imported, each with an appropriate status: either *Waiting for approval* or *Added*.
- For each *Waiting for approval* import job listed, there is a corresponding pull request adding the **catalog-info.yaml** file in the corresponding repository.

8.3. MANAGING THE ADDED REPOSITORIES

You can oversee and manage the repositories that are imported to the Developer Hub.

Prerequisites

- You have imported GitHub repositories.

Procedure

1. Click **Bulk Import** in the left sidebar to display all the current repositories that are being tracked as Import jobs, along with their status.

   Added

   The repository is added to the Developer Hub catalog after the import pull request is merged or if the repository already contained a **catalog-info.yaml** file during the bulk import. Note that it may take a few minutes for the entities to be available in the catalog.

   Waiting for approval

   There is an open pull request adding a **catalog-info.yaml** file to the repository. You can:

   - Click the **pencil icon** on the right to see details about the pull request or edit the pull request content right from Developer Hub.
   - Delete the Import job, this action closes the import PR as well.
   - To transition the Import job to the *Added* state, merge the import pull request from the Git repository.

   Empty

   Developer Hub is unable to determine the import job status because the repository is imported from other sources but does not have a **catalog-info.yaml** file and lacks any import pull request adding it.

NOTE

- After an import pull request is merged, the import status is marked as *Added* in the list of Added Repositories, but it might take a few seconds for the corresponding entities to appear in the Developer Hub Catalog.
- A location added through other sources (like statically in an **app-config.yaml** file, dynamically when enabling GitHub discovery, or registered manually using the "Register an existing component" page) might show up in the Bulk Import list of Added Repositories if the following conditions are met:
  - The target repository is accessible from the configured GitHub integrations.
  - The location URL points to a **catalog-info.yaml** file at the root of the repository default branch.

8.4. UNDERSTANDING THE BULK IMPORT AUDIT LOGS

The Bulk Import backend plugin adds the following events to the Developer Hub audit logs. See Audit Logs in Red Hat Developer Hub for more information on how to configure and view audit logs.

Bulk Import Events:

BulkImportUnknownEndpoint

Tracks requests to unknown endpoints.

BulkImportPing

Tracks **GET** requests to the **/ping** endpoint, which allows us to make sure the bulk import backend is up and running.

BulkImportFindAllOrganizations

Tracks **GET** requests to the **/organizations** endpoint, which returns the list of organizations accessible from all configured GitHub Integrations.

BulkImportFindRepositoriesByOrganization

Tracks **GET** requests to the **/organizations/:orgName/repositories** endpoint, which returns the list of repositories for the specified organization (accessible from any of the configured GitHub Integrations).

BulkImportFindAllRepositories

Tracks GET requests to the /**repositories** endpoint, which returns the list of repositories accessible from all configured GitHub Integrations.

BulkImportFindAllImports

Tracks **GET** requests to the **/imports** endpoint, which returns the list of existing import jobs along with their statuses.

BulkImportCreateImportJobs

Tracks **POST** requests to the **/imports** endpoint, which allows to submit requests to bulk-import one or many repositories into the Developer Hub catalog, by eventually creating import pull requests in the target repositories.

BulkImportFindImportStatusByRepo

Tracks **GET** requests to the **/import/by-repo** endpoint, which fetches details about the import job for the specified repository.

BulkImportDeleteImportByRepo

Tracks **DELETE** requests to the **/import/by-repo** endpoint, which deletes any existing import job for the specified repository, by closing any open import pull request that could have been created.

Example bulk import audit logs

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

CHAPTER 9. SERVICENOW CUSTOM ACTIONS IN RED HAT DEVELOPER HUB

IMPORTANT

These features are for Technology Preview only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend using them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information on Red Hat Technology Preview features, see Technology Preview Features Scope.

In Red Hat Developer Hub, you can access ServiceNow custom actions (custom actions) for fetching and registering resources in the catalog.

The custom actions in Developer Hub enable you to facilitate and automate the management of records. Using the custom actions, you can perform the following actions:

- Create, update, or delete a record
- Retrieve information about a single record or multiple records

9.1. ENABLING SERVICENOW CUSTOM ACTIONS PLUGIN IN RED HAT DEVELOPER HUB

In Red Hat Developer Hub, the ServiceNow custom actions are provided as a pre-loaded plugin, which is disabled by default. You can enable the custom actions plugin using the following procedure.

Prerequisites

- Red Hat Developer Hub is installed and running. For more information about installing the Developer Hub, see Installing Red Hat Developer Hub on OpenShift Container Platform with the Helm chart.
- You have created a project in the Developer Hub.

Procedure

1. To activate the custom actions plugin, add a **package** with plugin name and update the **disabled** field in your Helm chart as follows:

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

NOTE

The default configuration for a plugin is extracted from the **dynamic-plugins.default.yaml** file, however, you can use a **pluginConfig** entry to override the default configuration.

2. Set the following variables in the Helm chart to access the custom actions:

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

9.2. SUPPORTED SERVICENOW CUSTOM ACTIONS IN RED HAT DEVELOPER HUB

The ServiceNow custom actions enable you to manage records in the Red Hat Developer Hub. The custom actions support the following HTTP methods for API requests:

- **GET**: Retrieves specified information from a specified resource endpoint
- **POST**: Creates or updates a resource
- **PUT**: Modify a resource
- **PATCH**: Updates a resource
- **DELETE**: Deletes a resource

[GET] servicenow:now:table:retrieveRecord

Retrieves information of a specified record from a table in the Developer Hub.

Table 9.1. Input parameters

Name	Type	Requirement	Description
tableName	**string**	Required	Name of the table to retrieve the record from
sysId	**string**	Required	Unique identifier of the record to retrieve
sysparmDisplayValue	**enum("true", "false", "all")**	Optional	Returns field display values such as **true**, actual values as **false**, or both. The default value is **false**.

Name	Type	Requirement	Description
sysparmExcludeReferenceLink	**boolean**	Optional	Set as **true** to exclude Table API links for reference fields. The default value is **false**.
sysparmFields	**string[]**	Optional	Array of fields to return in the response
sysparmView	**string**	Optional	Renders the response according to the specified UI view. You can override this parameter using **sysparm_fields**.
sysparmQueryNoDomain	**boolean**	Optional	Set as **true** to access data across domains if authorized. The default value is **false**.

Table 9.2. Output parameters

Name	Type	Description
result	**Record<PropertyKey, unknown>**	The response body of the request

[GET] servicenow:now:table:retrieveRecords

Retrieves information about multiple records from a table in the Developer Hub.

Table 9.3. Input parameters

Name	Type	Requirement	Description
tableName	**string**	Required	Name of the table to retrieve the records from
sysparamQuery	**string**	Optional	Encoded query string used to filter the results
sysparmDisplayValue	**enum("true", "false", "all")**	Optional	Returns field display values such as **true**, actual values as **false**, or both. The default value is **false**.

Name	Type	Requirement	Description
sysparmExcludeReferenceLink	**boolean**	Optional	Set as **true** to exclude Table API links for reference fields. The default value is **false**.
sysparmSuppressPaginationHeader	**boolean**	Optional	Set as **true** to suppress pagination header. The default value is **false**.
sysparmFields	**string[]**	Optional	Array of fields to return in the response
sysparmLimit	**int**	Optional	Maximum number of results returned per page. The default value is **10,000**.
sysparmView	**string**	Optional	Renders the response according to the specified UI view. You can override this parameter using **sysparm_fields**.
sysparmQueryCategory	**string**	Optional	Name of the query category to use for queries
sysparmQueryNoDomain	**boolean**	Optional	Set as **true** to access data across domains if authorized. The default value is **false**.
sysparmNoCount	**boolean**	Optional	Does not execute a select count(*) on the table. The default value is **false**.

Table 9.4. Output parameters

Name	Type	Description
result	**Record<PropertyKey, unknown>**	The response body of the request

[POST] servicenow:now:table:createRecord

Creates a record in a table in the Developer Hub.

Table 9.5. Input parameters

Name	Type	Requirement	Description
tableName	**string**	Required	Name of the table to save the record in
requestBody	**Record<PropertyKey, unknown>**	Optional	Field name and associated value for each parameter to define in the specified record
sysparmDisplayValue	**enum("true", "false", "all")**	Optional	Returns field display values such as **true**, actual values as **false**, or both. The default value is **false**.
sysparmExcludeReferenceLink	**boolean**	Optional	Set as **true** to exclude Table API links for reference fields. The default value is **false**.
sysparmFields	**string[]**	Optional	Array of fields to return in the response
sysparmInputDisplayValue	**boolean**	Optional	Set field values using their display value such as **true** or actual value as **false**. The default value is **false**.
sysparmSuppressAutoSysField	**boolean**	Optional	Set as **true** to suppress auto-generation of system fields. The default value is **false**.
sysparmView	**string**	Optional	Renders the response according to the specified UI view. You can override this parameter using **sysparm_fields**.

Table 9.6. Output parameters

Name	Type	Description
result	**Record<PropertyKey, unknown>**	The response body of the request

[PUT] servicenow:now:table:modifyRecord

Modifies a record in a table in the Developer Hub.

Table 9.7. Input parameters

Name	Type	Requirement	Description
tableName	**string**	Required	Name of the table to modify the record from
sysId	**string**	Required	Unique identifier of the record to modify
requestBody	**Record<PropertyKey, unknown>**	Optional	Field name and associated value for each parameter to define in the specified record
sysparmDisplayValue	**enum("true", "false", "all")**	Optional	Returns field display values such as **true**, actual values as **false**, or both. The default value is **false**.
sysparmExcludeReferenceLink	**boolean**	Optional	Set as **true** to exclude Table API links for reference fields. The default value is **false**.
sysparmFields	**string[]**	Optional	Array of fields to return in the response
sysparmInputDisplayValue	**boolean**	Optional	Set field values using their display value such as **true** or actual value as**false**. The default value is **false**.
sysparmSuppressAutoSysField	**boolean**	Optional	Set as **true** to suppress auto-generation of system fields. The default value is **false**.
sysparmView	**string**	Optional	Renders the response according to the specified UI view. You can override this parameter using **sysparm_fields**.
sysparmQueryNoDomain	**boolean**	Optional	Set as **true** to access data across domains if authorized. The default value is **false**.

Table 9.8. Output parameters

Name	Type	Description
result	**Record<PropertyKey, unknown>**	The response body of the request

[PATCH] servicenow:now:table:updateRecord

Updates a record in a table in the Developer Hub.

Table 9.9. Input parameters

Name	Type	Requirement	Description
tableName	**string**	Required	Name of the table to update the record in
sysId	**string**	Required	Unique identifier of the record to update
requestBody	**Record<PropertyKey, unknown>**	Optional	Field name and associated value for each parameter to define in the specified record
sysparmDisplayValue	**enum("true", "false", "all")**	Optional	Returns field display values such as **true**, actual values as **false**, or both. The default value is **false**.
sysparmExcludeReferenceLink	**boolean**	Optional	Set as **true** to exclude Table API links for reference fields. The default value is **false**.
sysparmFields	**string[]**	Optional	Array of fields to return in the response
sysparmInputDisplayValue	**boolean**	Optional	Set field values using their display value such as **true** or actual value as**false**. The default value is **false**.
sysparmSuppressAutoSysField	**boolean**	Optional	Set as **true** to suppress auto-generation of system fields. The default value is **false**.
sysparmView	**string**	Optional	Renders the response according to the specified UI view. You can override this parameter using **sysparm_fields**.
sysparmQueryNoDomain	**boolean**	Optional	Set as **true** to access data across domains if authorized. The default value is **false**.

Table 9.10. Output parameters

Name	Type	Description
result	**Record<PropertyKey, unknown>**	The response body of the request

[DELETE] servicenow:now:table:deleteRecord

Deletes a record from a table in the Developer Hub.

Table 9.11. Input parameters

Name	Type	Requirement	Description
tableName	**string**	Required	Name of the table to delete the record from
sysId	**string**	Required	Unique identifier of the record to delete
sysparmQueryNoDomain	**boolean**	Optional	Set as **true** to access data across domains if authorized. The default value is **false**.

CHAPTER 10. KUBERNETES CUSTOM ACTIONS IN RED HAT DEVELOPER HUB

With Kubernetes custom actions, you can create and manage Kubernetes resources.

The Kubernetes custom actions plugin is preinstalled and disabled on a Developer Hub instance by default. You can disable or enable the Kubernetes custom actions plugin, and change other parameters, by configuring the Red Hat Developer Hub Helm chart.

NOTE

Kubernetes scaffolder actions and Kubernetes custom actions refer to the same concept throughout this documentation.

10.1. ENABLING KUBERNETES CUSTOM ACTIONS PLUGIN IN RED HAT DEVELOPER HUB

In Red Hat Developer Hub, the Kubernetes custom actions are provided as a preinstalled plugin, which is disabled by default. You can enable the Kubernetes custom actions plugin by updating the **disabled** key value in your Helm chart.

Prerequisites

- You have installed Red Hat Developer Hub with the Helm chart.

Procedure

To enable the Kubernetes custom actions plugin, complete the following step:

- In your Helm chart, add a **package** with the Kubernetes custom action plugin name and update the **disabled** field. For example:

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

NOTE

The default configuration for a plugin is extracted from the **dynamic-plugins.default.yaml** file, however, you can use a **pluginConfig** entry to override the default configuration.

10.2. USING KUBERNETES CUSTOM ACTIONS PLUGIN IN RED HAT DEVELOPER HUB

In Red Hat Developer Hub, the Kubernetes custom actions enable you to run template actions for Kubernetes.

Procedure

- To use a Kubernetes custom action in your custom template, add the following Kubernetes actions to your template:

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

Additional resource

- Configuring templates.

10.3. CREATING A TEMPLATE USING KUBERNETES CUSTOM ACTIONS IN RED HAT DEVELOPER HUB

You can create a template by defining a **Template** object as a YAML file.

The **Template** object describes the template and its metadata. It also contains required input variables and a list of actions that are executed by the scaffolding service.

+

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

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

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

10.3.1. Supported Kubernetes custom actions in Red Hat Developer Hub

In Red Hat Developer Hub, you can use custom Kubernetes actions in scaffolder templates.

Custom Kubernetes scaffolder actions

Action: kubernetes:create-namespace

Creates a namespace for the Kubernetes cluster in the Developer Hub.

Parameter name	Type	Requirement	Description	Example
namespace	**string**	Required	Name of the Kubernetes namespace	**my-rhdh-project**
clusterRef	**string**	Required only if **url** is not defined. You cannot specify both **url** and **clusterRef**.	Cluster resource entity reference from the catalog	**bar**
url	**string**	Required only if **clusterRef** is not defined. You cannot specify both **url** and **clusterRef**.	API url of the Kubernetes cluster	**https://api.foo.redhat.com:6443**
token	**String**	Required	Kubernetes API bearer token used for authentication	
skipTLSVerify	**boolean**	Optional	If true, certificate verification is skipped	false
caData	**string**	Optional	Base64 encoded certificate data	
label	**string**	Optional	Labels applied to the namespace	app.io/type=ns; app.io/managed-by=org;

CHAPTER 11. OVERRIDING CORE BACKEND SERVICE CONFIGURATION

The Red Hat Developer Hub (RHDH) backend platform consists of a number of core services that are well encapsulated. The RHDH backend installs these default core services statically during initialization.

Customize a core service by installing it as a **BackendFeature** by using the dynamic plugin functionality.

Procedure

1. Configure Developer Hub to allow a core service override, by setting the corresponding core service ID environment variable to **true** in the Developer Hub **app-config.yaml** configuration file.

Table 11.1. Environment variables and core service IDs

Variable	Overrides the related service
ENABLE_CORE_AUTH_OVERRIDE	**core.auth**
ENABLE_CORE_CACHE_OVERRIDE	**core.cache**
ENABLE_CORE_ROOTCONFIG_OVERRIDE	**core.rootConfig**
ENABLE_CORE_DATABASE_OVERRIDE	**core.database**
ENABLE_CORE_DISCOVERY_OVERRIDE	**core.discovery**
ENABLE_CORE_HTTPAUTH_OVERRIDE	**core.httpAuth**
ENABLE_CORE_HTTPROUTER_OVERRIDE	**core.httpRouter**
ENABLE_CORE_LIFECYCLE_OVERRIDE	**core.lifecycle**
ENABLE_CORE_LOGGER_OVERRIDE	**core.logger**
ENABLE_CORE_PERMISSIONS_OVERRIDE	**core.permissions**
ENABLE_CORE_ROOTHEALTH_OVERRIDE	**core.rootHealth**
ENABLE_CORE_ROOTHTTPROUTER_OVERRIDE	**core.rootHttpRouter**

Variable	Overrides the related service
ENABLE_CORE_ROOTLIFECYCLE_OVERRIDE	**core.rootLifecycle**
ENABLE_CORE_SCHEDULER_OVERRIDE	**core.scheduler**
ENABLE_CORE_USERINFO_OVERRIDE	**core.userInfo**
ENABLE_CORE_URLREADER_OVERRIDE	**core.urlReader**
ENABLE_EVENTS_SERVICE_OVERRIDE	**events.service**

2. Install your custom core service as a **BackendFeature** as shown in the following example:

Example of a **BackendFeature** middleware function to handle incoming **HTTP** requests

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

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

+ In the previous example, as the **BackendFeature** overrides the default implementation of the HTTP router service, you must set the **ENABLE_CORE_ROOTHTTPROUTER_OVERRIDE** environment variable to **true** so that the Developer Hub does not install the default implementation automatically.