



Red Hat Developer Hub 1.1

Administration guide for Red Hat Developer Hub

Red Hat Developer Hub 1.1 Administration guide for Red Hat Developer Hub

Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Red Hat Developer Hub is a developer platform for building developer portals. This document provides an overview of Red Hat Developer Hub and explains how to install the product.

Table of Contents

PREFACE	4
RED HAT DEVELOPER HUB SUPPORT	5
CHAPTER 1. INSTALLING THE RED HAT DEVELOPER HUB OPERATOR	6
CHAPTER 2. DEPLOYING RED HAT DEVELOPER HUB ON OPENSIFT CONTAINER PLATFORM	8
2.1. DEPLOYING RED HAT DEVELOPER HUB ON OPENSIFT CONTAINER PLATFORM USING HELM CHART	8
2.1.1. Installing Red Hat Developer Hub using the Helm Chart in an air-gapped environment	10
2.2. DEPLOYING RED HAT DEVELOPER HUB ON OPENSIFT CONTAINER PLATFORM USING THE OPERATOR	12
2.2.1. Configuring the Developer Hub Custom Resource	13
2.2.1.1. Adding a custom application configuration file to OpenShift Container Platform	13
2.2.2. Configuring dynamic plugins with the Red Hat Developer Hub Operator	16
2.2.3. Installing Red Hat Developer Hub using the Operator in an air-gapped environment	18
CHAPTER 3. RED HAT DEVELOPER HUB INTEGRATION WITH AMAZON WEB SERVICES (AWS)	21
3.1. DEPLOYING RED HAT DEVELOPER HUB IN ELASTIC KUBERNETES SERVICE (EKS) USING HELM CHART	21
3.2. DEPLOYING RED HAT DEVELOPER HUB ON ELASTIC KUBERNETES SERVICE (EKS) USING THE OPERATOR	23
3.2.1. Installing the Red Hat Developer Hub Operator with the OLM framework	23
3.2.2. Installing the Red Hat Developer Hub Operator without the OLM framework	26
3.2.3. Installing the Developer Hub instance in EKS	28
3.3. MONITORING AND LOGGING WITH AMAZON WEB SERVICES (AWS) IN RED HAT DEVELOPER HUB	31
3.3.1. Monitoring with Amazon Prometheus	31
3.3.1.1. Configuring annotations for monitoring	32
3.3.2. Logging with Amazon CloudWatch logs	33
3.3.2.1. Configuring the application log level	33
3.3.2.2. Retrieving logs from Amazon CloudWatch	33
3.4. USING AMAZON COGNITO AS AN AUTHENTICATION PROVIDER IN RED HAT DEVELOPER HUB	34
CHAPTER 4. RED HAT DEVELOPER HUB INTEGRATION WITH MICROSOFT AZURE KUBERNETES SERVICE (AKS)	39
4.1. DEPLOYING RED HAT DEVELOPER HUB ON AZURE KUBERNETES SERVICE (AKS)	39
4.1.1. Deploying the Red Hat Developer Hub on Azure Kubernetes Service (AKS) using the Helm chart	40
4.1.2. Deploying the Red Hat Developer Hub on Azure Kubernetes Service (AKS) using the Operator	42
4.2. MONITORING AND LOGGING WITH AZURE KUBERNETES SERVICES (AKS) IN RED HAT DEVELOPER HUB	44
4.2.1. Viewing logs with Azure Kubernetes Services (AKS)	45
4.3. USING MICROSOFT AZURE AS AN AUTHENTICATION PROVIDER IN RED HAT DEVELOPER HUB	46
4.3.1. Using Microsoft Azure as an authentication provider in Helm deployment	46
4.3.2. Using Microsoft Azure as an authentication provider in Operator-backed deployment	47
CHAPTER 5. ROLE-BASED ACCESS CONTROL (RBAC) IN RED HAT DEVELOPER HUB	49
5.1. PERMISSION POLICIES CONFIGURATION	49
5.1.1. Configuration of permission policies administrators	49
5.1.2. Configuration of permission policies defined in an external file	49
5.1.2.1. Mounting policy.csv file to the Developer Hub Helm Chart	50
5.1.3. Permission policies in Red Hat Developer Hub	51
5.2. MANAGING ROLE-BASED ACCESS CONTROLS (RBAC) USING THE RED HAT DEVELOPER HUB WEB UI	53
5.2.1. Creating a role in the Red Hat Developer Hub Web UI	53

5.2.2. Editing a role in the Red Hat Developer Hub Web UI	54
5.2.3. Deleting a role in the Red Hat Developer Hub Web UI	54
5.3. ROLE-BASED ACCESS CONTROL (RBAC) REST API	55
5.3.1. Sending requests with the RBAC REST API using a REST client or curl utility	57
5.3.2. Supported RBAC REST API endpoints	58
5.3.2.1. Permission policies	58
5.3.2.2. Roles	62
CHAPTER 6. DYNAMIC PLUGIN INSTALLATION	65
6.1. VIEWING INSTALLED PLUGINS	65
6.2. PREINSTALLED DYNAMIC PLUGINS	65
6.2.1. Preinstalled dynamic plugin descriptions and details	65
6.3. INSTALLATION OF DYNAMIC PLUGINS USING THE HELM CHART	87
6.3.1. Obtaining the integrity checksum	88
6.3.2. Example Helm chart configurations for dynamic plugin installations	88
6.3.3. Installing external dynamic plugins using a Helm chart	89
6.4. INSTALLING EXTERNAL PLUGINS IN AN AIR-GAPPED ENVIRONMENT	90
6.5. USING A CUSTOM NPM REGISTRY FOR DYNAMIC PLUGIN PACKAGES	90
6.6. BASIC CONFIGURATION OF DYNAMIC PLUGINS	90
6.7. INSTALLATION AND CONFIGURATION OF ANSIBLE AUTOMATION PLATFORM	91
6.7.1. For administrators	91
6.7.1.1. Installing and configuring the AAP Backend plugin	91
6.7.1.2. Log lines for AAP Backend plugin troubleshoot	92
6.7.2. For users	92
6.7.2.1. Accessing templates from AAP in Developer Hub	92
6.8. INSTALLATION AND CONFIGURATION OF KEYCLOAK	94
6.8.1. For administrators	94
6.8.1.1. Installation	94
6.8.1.2. Basic configuration	94
6.8.1.3. Advanced configuration	95
6.8.1.4. Limitations	96
6.8.2. For users	97
6.8.2.1. Import of users and groups in Developer Hub using the Keycloak plugin	97
6.9. INSTALLATION AND CONFIGURATION OF NEXUS REPOSITORY MANAGER	99
6.9.1. For administrators	99
6.9.1.1. Installing and configuring the Nexus Repository Manager plugin	99
6.9.2. For users	100
6.9.2.1. Using the Nexus Repository Manager plugin in Developer Hub	100
6.10. INSTALLATION AND CONFIGURATION OF TEKTON	101
6.10.1. For administrators	101
6.10.1.1. Installation	101
6.10.2. For users	103
6.10.2.1. Using the Tekton plugin in RHDH	104
CHAPTER 7. MANAGING TEMPLATES	105
7.1. CREATING A TEMPLATE BY USING THE TEMPLATE EDITOR	105
7.2. CREATING A TEMPLATE AS A YAML FILE	106
7.3. IMPORTING AN EXISTING TEMPLATE TO RED HAT DEVELOPER HUB	108

PREFACE

The Red Hat Developer Hub is an enterprise-grade, open developer platform that you can use to build developer portals. This platform contains a supported and opinionated framework that helps reduce the friction and frustration of developers while boosting their productivity.

RED HAT DEVELOPER HUB SUPPORT

If you experience difficulty with a procedure described in this documentation, visit the [Red Hat Customer Portal](#). You can use the Red Hat Customer Portal for the following purposes:

- To search or browse through the Red Hat Knowledgebase of technical support articles about Red Hat products.
- To create a [support case](#) for Red Hat Global Support Services (GSS). For support case creation, select **Red Hat Developer Hub** as the product and select the appropriate product version.

CHAPTER 1. INSTALLING THE RED HAT DEVELOPER HUB OPERATOR

As an administrator, you can install the Red Hat Developer Hub Operator. Authorized users can use the Operator to install Red Hat Developer Hub on the following platforms:

- Red Hat OpenShift Container Platform (RHOCP)
- Amazon Elastic Kubernetes Service (EKS)
- Microsoft Azure Kubernetes Service (AKS)

Prerequisites

- You are logged in as an administrator on the OpenShift Container Platform web console.
- You have configured the appropriate roles and permissions within your project to create an application. For more information, see the [Red Hat OpenShift documentation on Building applications](#).



NOTE

For enhanced security, deploy the Red Hat Developer Hub Operator in a dedicated default namespace such as **rdh-operator**. The cluster administrator can restrict other users' access to the Operator resources through role bindings or cluster role bindings.

Procedure

1. In the **Administrator** perspective of the OpenShift Container Platform web console, navigate to **Operators > OperatorHub**.
2. In the **Filter by keyword** box, enter Developer Hub and click the **Red Hat Developer Hub Operator** card.
3. On the **Red Hat Developer Hub Operator** page, click **Install**.
4. On the **Install Operator** page, use the **Update channel** drop-down menu to select the update channel that you want to use:
 - The **fast** channel provides y-stream (x.y) and z-stream (x.y.z) updates, for example, updating from version 1.1 to 1.2, or from 1.1.0 to 1.1.1.



IMPORTANT

The **fast** channel includes all of the updates available for a particular version. Any update might introduce unexpected changes in your Red Hat Developer Hub deployment. Check the release notes for details about any potentially breaking changes.

- The **fast-1.1** channel only provides z-stream updates, for example, updating from version 1.1.1 to 1.1.2. If you want to update the Red Hat Developer Hub y-version in the future, for example, updating from 1.1 to 1.2, you must switch to the **fast** channel manually.
5. On the **Install Operator** page, choose the **Update approval** strategy for the Operator:

- If you choose the **Automatic** option, the Operator is updated without requiring manual confirmation.
- If you choose the **Manual** option, a notification opens when a new update is released in the update channel. The update must be manually approved by an administrator before installation can begin.

6. Click **Install**.

Verification

- To view the installed Red Hat Developer Hub Operator, click **View Operator**.

Additional resources

- [Deploying Red Hat Developer Hub on OpenShift Container Platform using the Operator](#)
- [Installing from OperatorHub using the web console](#)

CHAPTER 2. DEPLOYING RED HAT DEVELOPER HUB ON OPENSIFT CONTAINER PLATFORM

You can install Red Hat Developer Hub on OpenShift Container Platform using one of the following methods:

- The Helm chart
- The Red Hat Developer Hub Operator

2.1. DEPLOYING RED HAT DEVELOPER HUB ON OPENSIFT CONTAINER PLATFORM USING HELM CHART

You can use a Helm chart in Red Hat OpenShift Container Platform to install Developer Hub, which is a flexible installation method.

Helm is a package manager on OpenShift Container Platform that provides the following features:

- Applies regular application updates using custom hooks
- Manages the installation of complex applications
- Provides charts that you can host on public and private servers
- Supports rolling back to previous application versions

The Red Hat Developer Hub Helm chart is available in the Helm catalog on OpenShift Dedicated and OpenShift Container Platform.

Prerequisites

- You are logged in to your OpenShift Container Platform account.
- A user with the OpenShift Container Platform **admin** role has configured the appropriate roles and permissions within your project to create an application. For more information about OpenShift Container Platform roles, see [Using RBAC to define and apply permissions](#) .
- You have created a project in OpenShift Container Platform. For more information about creating a project in OpenShift Container Platform, see [Red Hat OpenShift Container Platform documentation](#).

Procedure

1. From the **Developer** perspective on the Developer Hub web console, click **+Add**.
2. From the **Developer Catalog** panel, click **Helm Chart**.
3. In the **Filter by keyword** box, enter *Developer Hub* and click the **Red Hat Developer Hub** card.
4. From the Red Hat Developer Hub page, click **Create**.
5. From your cluster, copy the OpenShift Container Platform router host (for example: **apps.<clusterName>.com**).

- Select the radio button to configure the Developer Hub instance with either the form view or YAML view. The Form view is selected by default.

- Using **Form view**

- To configure the instance with the Form view, go to **Root Schema** → **global** → **Enable service authentication within Backstage instance** and paste your OpenShift Container Platform router host into the field on the form.

- Using **YAML view**

- To configure the instance with the YAML view, paste your OpenShift Container Platform router hostname in the **global.clusterRouterBase** parameter value as shown in the following example:

```
global:
  auth:
    backend:
      enabled: true
  clusterRouterBase: apps.<clusterName>.com
# other Red Hat Developer Hub Helm Chart configurations
```

- Edit the other values if needed.



NOTE

The information about the host is copied and can be accessed by the Developer Hub backend.

When an OpenShift Container Platform route is generated automatically, the host value for the route is inferred and the same host information is sent to the Developer Hub. Also, if the Developer Hub is present on a custom domain by setting the host manually using values, the custom host takes precedence.

- Click **Create** and wait for the database and Developer Hub to start.
- Click the **Open URL** icon to start using the Developer Hub platform.





NOTE

Your **developer-hub** pod might be in a **CrashLoopBackOff** state if the Developer Hub container cannot access the configuration files. This error is indicated by the following log:

```
Loaded config from app-config-from-configmap.yaml, env
...
2023-07-24T19:44:46.223Z auth info Configuring "database" as KeyStore provider
type=plugin
Backend failed to start up Error: Missing required config value at
'backend.database.client'
```

To resolve the error, verify the configuration files.

2.1.1. Installing Red Hat Developer Hub using the Helm Chart in an air-gapped environment

An air-gapped environment, also known as an air-gapped network or isolated network, ensures security by physically segregating the system or network. This isolation is established to prevent unauthorized access, data transfer, or communication between the air-gapped system and external sources.

You can install Red Hat Developer Hub in an air-gapped environment to ensure security and meet specific regulatory requirements.

To install Developer Hub in an air-gapped environment, you must have access to the **registry.redhat.io** and the registry for the air-gapped environment.

Prerequisites

- You have installed an Red Hat OpenShift Container Platform 4.12 or later.
- You have access to the **registry.redhat.io**.
- You have access to the Red Hat OpenShift Container Platform image registry of your cluster. For more information about exposing the image registry, see the Red Hat OpenShift Container Platform documentation about [Exposing the registry](#).
- You have installed the **oc** command line tool on your workstation.
- You have installed the **podman** command line tools on your workstation.
- You you have an account in [Red Hat Developer](#) portal.

Procedure

1. Log in to your OpenShift Container Platform account using the **oc** command line tool, by running the following command:

```
oc login -u <user> -p <password> https://api.<hostname>:6443
```

2. Log in to the OpenShift Container Platform image registry using the **podman** command line tool, by running the following command:

```
podman login -u kubeadmin -p $(oc whoami -t) default-route-openshift-image-registry.<hostname>
```



NOTE

You can run the following commands to get the full host name of the OpenShift Container Platform image registry, and then use the host name in a command to log in:

```
REGISTRY_HOST=$(oc get route default-route -n openshift-image-registry --template='{{ .spec.host }}')
```

```
podman login -u kubeadmin -p $(oc whoami -t) $REGISTRY_HOST
```

3. Log in to the **registry.redhat.io** in **podman** by running the following command:

```
podman login registry.redhat.io
```

For more information about registry authentication, see [Red Hat Container Registry Authentication](#).

4. Pull Developer Hub and PostgreSQL images from [Red Hat Image registry](#) to your workstation, by running the following commands:

```
podman pull registry.redhat.io/rhdh/rhdh-hub-rhel9:{product-chart-version}
```

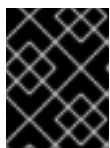
```
podman pull registry.redhat.io/rhel9/postgresql-15:latest
```

5. Push both images to the internal OpenShift Container Platform image registry by running the following commands:

```
podman push --remove-signatures registry.redhat.io/rhdh/rhdh-hub-rhel9:{product-chart-version} default-route-openshift-image-registry.<hostname>/<project_name>/rhdh-hub-rhel9:{product-chart-version}
```

```
podman push --remove-signatures registry.redhat.io/rhel9/postgresql-15:latest default-route-openshift-image-registry.<hostname>/<project_name>/postgresql-15:latest
```

For more information about pushing images directly to the OpenShift Container Platform image registry, see [How do I push an Image directly into the OpenShift 4 registry](#) .



IMPORTANT

If an x509 error occurs, verify that you have [installed the CA certificate used for OpenShift Container Platform routes on your system](#).

6. Use the following command to verify that both images are present in the internal OpenShift Container Platform registry:

```
oc get imagestream -n <project_name>
```

7. Enable local image lookup for both images by running the following commands:

```
oc set image-lookup postgresql-15
```

```
oc set image-lookup rhdh-hub-rhel9
```

8. Go to **YAML view** and update the **image** section for **backstage** and **postgresql** using the following values:

Example values for Developer Hub image

```
upstream:
  backstage:
    image:
      registry: ""
      repository: rhdh-hub-rhel9
      tag: latest
```

Example values for PostgreSQL image

```
upstream:
  postgresql:
    image:
      registry: ""
      repository: postgresql-15
      tag: latest
```

9. Install the Red Hat Developer Hub using Helm chart. For more information about installing Developer Hub, see [Section 2.1, "Deploying Red Hat Developer Hub on OpenShift Container Platform using Helm Chart"](#).

2.2. DEPLOYING RED HAT DEVELOPER HUB ON OPENSIFT CONTAINER PLATFORM USING THE OPERATOR

As a developer, you can deploy a Red Hat Developer Hub instance on OpenShift Container Platform by using the **Developer Catalog** in the Red Hat OpenShift Container Platform web console. This deployment method uses the Red Hat Developer Hub Operator.

Prerequisites

- A cluster administrator has installed the Red Hat Developer Hub Operator. For more information, see [Installing the Red Hat Developer Hub Operator](#).

Procedure

1. Create a project in OpenShift Container Platform for your Red Hat Developer Hub instance, or select an existing project.

TIP

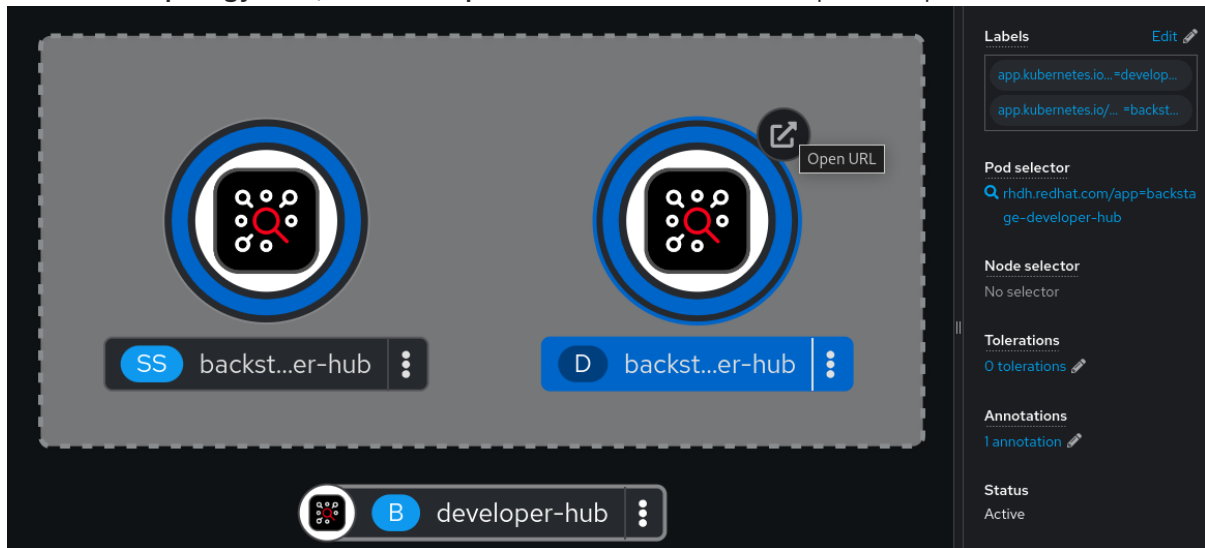
For more information about creating a project in OpenShift Container Platform, see [Creating a project by using the web console](#) in the Red Hat OpenShift Container Platform documentation.

2. From the **Developer** perspective on the OpenShift Container Platform web console, click **+Add**.
3. From the **Developer Catalog** panel, click **Operator Backed**.
4. In the **Filter by keyword** box, enter *Developer Hub* and click the **Red Hat Developer Hub** card.
5. Click **Create**.
6. Add [custom configurations](#) for the Red Hat Developer Hub instance.
7. On the **Create Backstage** page, click **Create**

Verification

After the pods are ready, you can access the Red Hat Developer Hub platform by opening the URL.

1. Confirm that the pods are ready by clicking the pod in the **Topology** view and confirming the **Status** in the **Details** panel. The pod status is **Active** when the pod is ready.
2. From the **Topology** view, click the **Open URL** icon on the Developer Hub pod.



Additional resources

- [OpenShift Container Platform - Building applications overview](#)

2.2.1. Configuring the Developer Hub Custom Resource

Updates to the Backstage custom resource (CR) are automatically handled by the Red Hat Developer Hub Operator. However, updates to resources referenced by the CR, such as ConfigMaps or Secrets, are not updated automatically unless the CR itself is updated. If you want to update resources referenced by the CR, then you must manually delete the Backstage Deployment so that the Operator can re-create it with the updated resources.

2.2.1.1. Adding a custom application configuration file to OpenShift Container Platform

To change the configuration of your Red Hat Developer Hub instance, you must do the following:

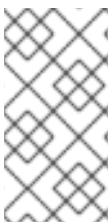
- Add a custom application configuration file to OpenShift Container Platform and reference it in the Custom Resource (CR). In OpenShift Container Platform, you can use the following example as a base template to create a ConfigMap such as **app-config-rhdh.yaml**:

```

kind: ConfigMap
apiVersion: v1
metadata:
  name: app-config-rhdh
data:
  "app-config-rhdh.yaml": |
    app:
      title: Red Hat Developer Hub
      baseUrl: https://backstage-developer-hub-my-ns.apps.ci-ln-vtkzr22-72292.origin-ci-int-
gce.dev.rhcloud.com
      backend:
        auth:
          keys:
            - secret: "${BACKEND_SECRET}"
        baseUrl: https://backstage-backstage-sample-my-ns.apps.ci-ln-vtkzr22-72292.origin-ci-
int-gce.dev.rhcloud.com
        cors:
          origin: https://backstage-backstage-sample-my-ns.apps.ci-ln-vtkzr22-72292.origin-ci-int-
gce.dev.rhcloud.com

```

- Use the mandatory backend auth key for Red Hat Developer Hub to reference an environment variable defined in an OpenShift Container Platform Secret.
- Set the external URL of your Red Hat Developer Hub instance in the **app.baseUrl**, **backend.baseUrl** and **backend.cors.origin** fields of the application configuration. By default, the URL is similar to the following example: **https://backstage-
<CUSTOM_RESOURCE_NAME>-<NAMESPACE_NAME>.
<OPENSIFT_INGRESS_DOMAIN>**;
 - You can use the **oc get ingresses.config/cluster -o jsonpath='{.spec.domain}'** command to display your ingress domain. If you want to use a different host or sub-domain, customize the **Custom Resource spec.application.route field** and adjust the application configuration accordingly.



NOTE

You are responsible for protecting your Red Hat Developer Hub installation from external and unauthorized access. Manage the backend auth key like any other secret. Meet strong password requirements, do not expose it in any configuration files, and only inject it into configuration files as an environment variable.

Prerequisites

- You have an active Red Hat OpenShift Container Platform account.

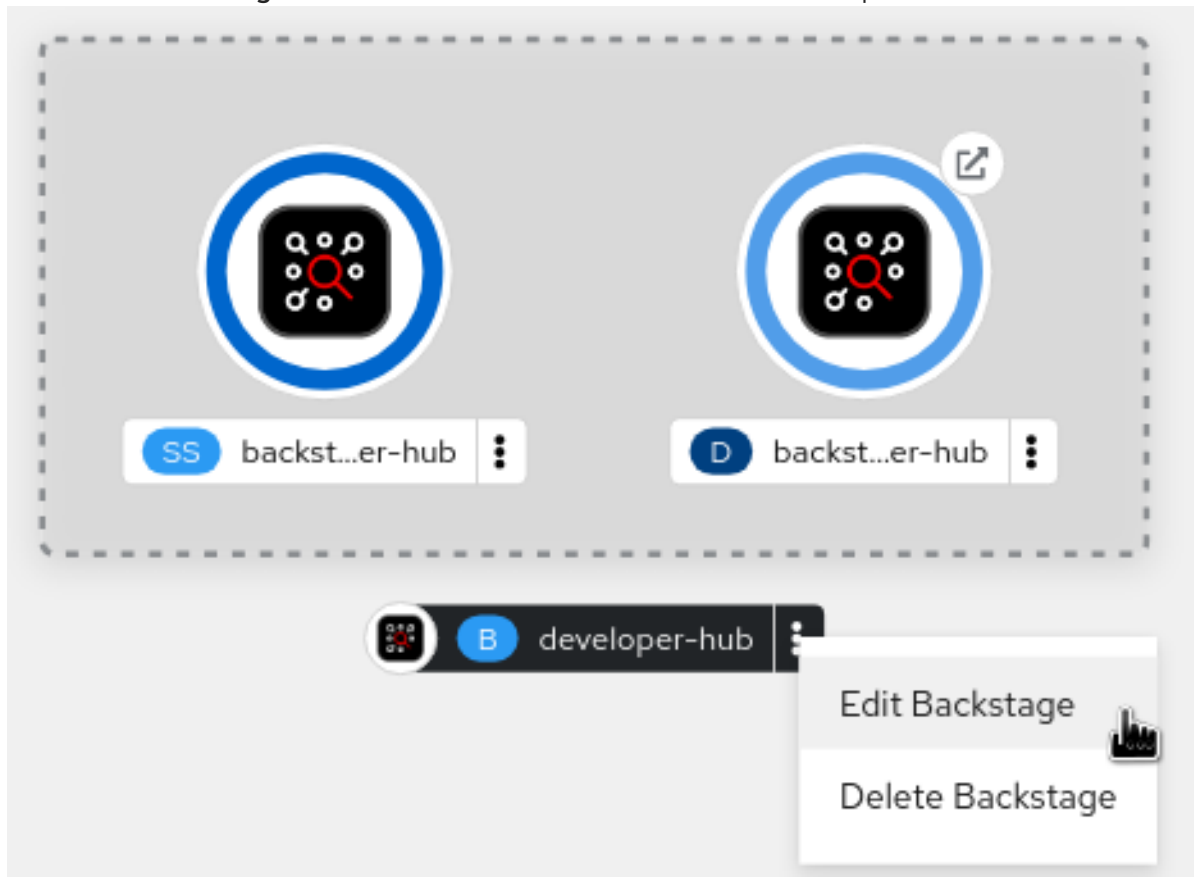
Procedure

1. From the **Developer** perspective, select the **ConfigMaps** tab.
2. Click **Create ConfigMap**.
3. Select the **YAML view** option in **Configure via** and make the changes to the file, if necessary.
4. Click **Create**.

5. Select the **Secrets** tab.
6. Click **Create Key/value Secret**
7. Name the secret **secrets-rhdh**.
8. Add a key named **BACKEND_SECRET** and a base64 encoded string as a value. Use a unique value for each Red Hat Developer Hub instance. For example, you can use the following command to generate a key from your terminal:

```
node -p 'require("crypto").randomBytes(24).toString("base64")'
```

9. Click **Create**.
10. Select the **Topology** view.
11. Click the overflow menu for the Red Hat Developer Hub instance that you want to use and select **Edit Backstage** to load the YAML view of the Red Hat Developer Hub instance.



12. Add the **spec.application.appConfig.configMaps** and **spec.application.extraEnvs.secrets** fields to the custom resource. For example:

```
spec:
  application:
    appConfig:
      mountPath: /opt/app-root/src
      configMaps:
        - name: app-config-rhdh
    extraEnvs:
      secrets:
        - name: secrets-rhdh
```

```

extraFiles:
  mountPath: /opt/app-root/src
replicas: 1
route:
  enabled: true
database:
  enableLocalDb: true

```

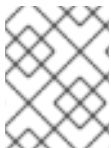
13. Click **Save**.
14. Navigate back to the **Topology** view and wait for the Red Hat Developer Hub pod to start.
15. Click the **Open URL** icon to start using the Red Hat Developer Hub platform with the new configuration changes.

Additional resources

- For more information about roles and responsibilities in Developer Hub, see [Role-Based Access Control \(RBAC\) in Red Hat Developer Hub](#) in the Administration Guide for Red Hat Developer Hub.

2.2.2. Configuring dynamic plugins with the Red Hat Developer Hub Operator

You can store the configuration for dynamic plugins in a **ConfigMap** object that your **Backstage** custom resource (CR) can reference.



NOTE

If the **pluginConfig** field references environment variables, you must define the variables in your **secrets-rhdh** secret.

Procedure

1. From the OpenShift Container Platform web console, select the **ConfigMaps** tab.
2. Click **Create ConfigMap**.
3. From the **Create ConfigMap** page, select the **YAML view** option in **Configure via** and edit the file, if needed.

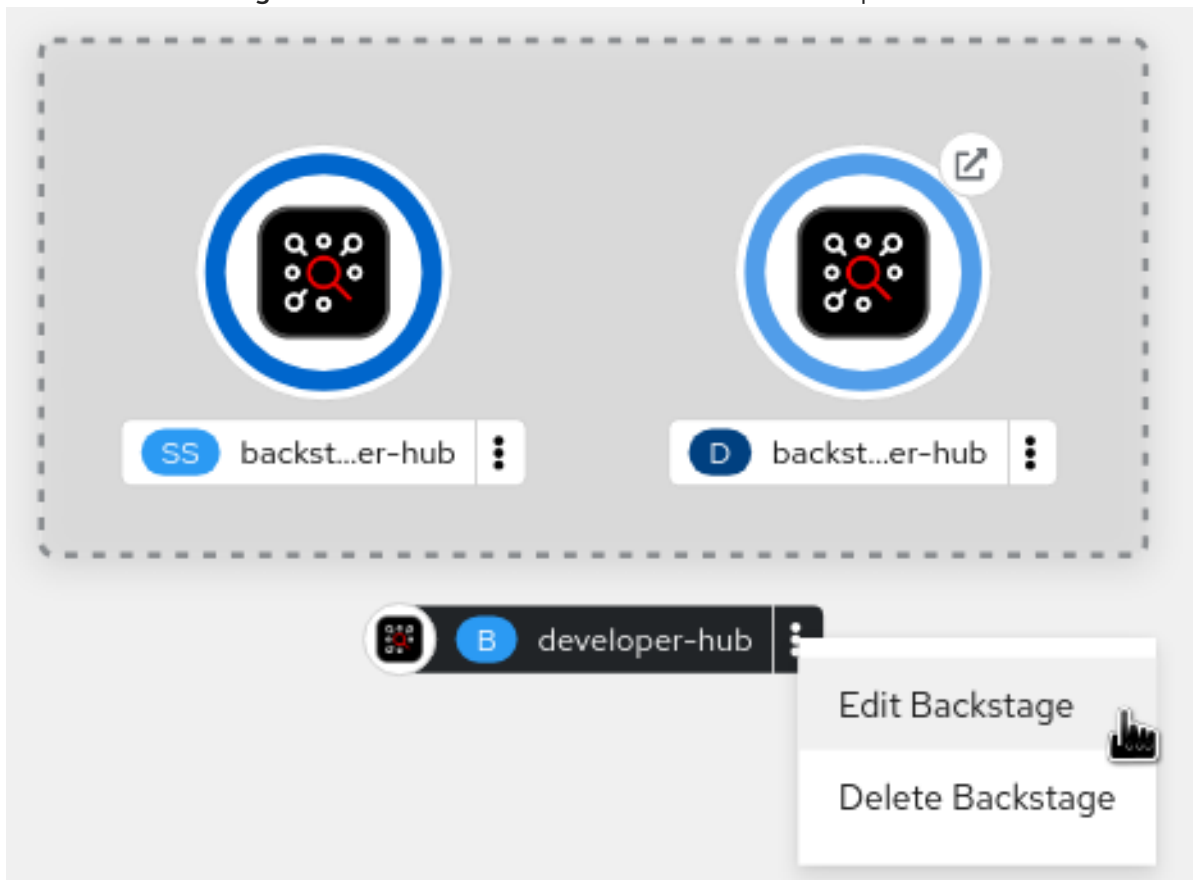
Example ConfigMap object using the GitHub dynamic plugin

```

kind: ConfigMap
apiVersion: v1
metadata:
  name: dynamic-plugins-rhdh
data:
  dynamic-plugins.yaml: |
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: './dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-
dynamic'
      disabled: false
      pluginConfig: {}

```

4. Click **Create**.
5. Go to the **Topology** tab.
6. Click on the overflow menu for the Red Hat Developer Hub instance that you want to use and select **Edit Backstage** to load the YAML view of the Red Hat Developer Hub instance.



7. Add the **dynamicPluginsConfigMapName** field to your **Backstage** CR. For example:

```

apiVersion: rhdh.redhat.com/v1alpha1
kind: Backstage
metadata:
  name: my-rhdh
spec:
  application:
# ...
    dynamicPluginsConfigMapName: dynamic-plugins-rhdh
# ...

```

8. Click **Save**.
9. Navigate back to the **Topology** view and wait for the Red Hat Developer Hub pod to start.
10. Click the **Open URL** icon to start using the Red Hat Developer Hub platform with the new configuration changes.

Verification

- Ensure that the dynamic plugins configuration has been loaded, by appending **/api/dynamic-plugins-info/loaded-plugins** to your Red Hat Developer Hub root URL and checking the list of plugins:

Example list of plugins

```
[
  {
    "name": "backstage-plugin-catalog-backend-module-github-dynamic",
    "version": "0.5.2",
    "platform": "node",
    "role": "backend-plugin-module"
  },
  {
    "name": "backstage-plugin-techdocs",
    "version": "1.10.0",
    "role": "frontend-plugin",
    "platform": "web"
  },
  {
    "name": "backstage-plugin-techdocs-backend-dynamic",
    "version": "1.9.5",
    "platform": "node",
    "role": "backend-plugin"
  },
]
```

2.2.3. Installing Red Hat Developer Hub using the Operator in an air-gapped environment

On an OpenShift Container Platform cluster operating on a restricted network, public resources are not available. However, deploying the Red Hat Developer Hub Operator and running Developer Hub requires the following public resources:

- Operator images (bundle, operator, catalog)
- Operands images (RHDH, PostgreSQL)

To make these resources available, replace them with their equivalent resources in a mirror registry accessible to the OpenShift Container Platform cluster.

You can use a helper script that mirrors the necessary images and provides the necessary configuration to ensure those images will be used when installing the Red Hat Developer Hub Operator and creating Developer Hub instances.



NOTE

This script requires a target mirror registry which you should already have installed if your OpenShift Container Platform cluster is ready to operate on a restricted network. However, if you are preparing your cluster for disconnected usage, you can use the script to deploy a mirror registry in the cluster and use it for the mirroring process.

Prerequisites

- You have an active **oc** session with administrative permissions to the OpenShift Container Platform cluster. See [Getting started with the OpenShift CLI](#).
- You have an active **oc registry** session to the **registry.redhat.io** Red Hat Ecosystem Catalog. See [Red Hat Container Registry Authentication](#).
- The **opm** CLI tool is installed. See [Installing the opm CLI](#).
- The jq package is installed. See [Download jq](#).
- Podman is installed. See [Podman Installation Instructions](#).
- Skopeo version 1.14 or higher is installed. See [Installing Skopeo](#).
- If you already have a mirror registry for your cluster, an active Skopeo session with administrative access to this registry is required. See [Authenticating to a registry](#) and [Mirroring images for a disconnected installation](#).



NOTE

The internal OpenShift Container Platform cluster image registry cannot be used as a target mirror registry. See [About the mirror registry](#).

- If you prefer to create your own mirror registry, see [Creating a mirror registry with mirror registry for Red Hat OpenShift](#).
- If you do not already have a mirror registry, you can use the helper script to create one for you and you need the following additional prerequisites:
 - The cURL package is installed. For Red Hat Enterprise Linux, the curl command is available by installing the curl package. To use curl for other platforms, see the [cURL website](#).
 - The **htpasswd** command is available. For Red Hat Enterprise Linux, the **htpasswd** command is available by installing the **httpd-tools** package.

Procedure

1. Download and run the mirroring script to install a custom Operator catalog and mirror the related images: **prepare-restricted-environment.sh** ([source](#)).

```
curl -sSLO https://raw.githubusercontent.com/janus-idp/operator/1.1.x/.rhdh/scripts/prepare-restricted-environment.sh
```

```
# if you do not already have a target mirror registry
# and want the script to create one for you
# use the following example:
```

```
bash prepare-restricted-environment.sh \
  --prod_operator_index "registry.redhat.io/redhat/redhat-operator-index:v4.14" \
  --prod_operator_package_name "rhdh" \
  --prod_operator_bundle_name "rhdh-operator" \
  --prod_operator_version "v1.1.1"
```

```
# if you already have a target mirror registry
# use the following example:
```

```
bash prepare-restricted-environment.sh \
  --prod_operator_index "registry.redhat.io/redhat/redhat-operator-index:v4.14" \
```

```
--prod_operator_package_name "rhdh" \  
--prod_operator_bundle_name "rhdh-operator" \  
--prod_operator_version "v1.1.1" \  
--use_existing_mirror_registry "my_registry"
```



NOTE

The script can take several minutes to complete as it copies multiple images to the mirror registry.

CHAPTER 3. RED HAT DEVELOPER HUB INTEGRATION WITH AMAZON WEB SERVICES (AWS)

You can integrate your Red Hat Developer Hub application with Amazon Web Services (AWS), which can help you streamline your workflows within the AWS ecosystem. Integrating the Developer Hub resources with AWS provides access to a comprehensive suite of tools, services, and solutions.

The integration with AWS requires the deployment of Developer Hub in Elastic Kubernetes Service (EKS) using one of the following methods:

- The Helm chart
- The Red Hat Developer Hub Operator

3.1. DEPLOYING RED HAT DEVELOPER HUB IN ELASTIC KUBERNETES SERVICE (EKS) USING HELM CHART

When you deploy Developer Hub in Elastic Kubernetes Service (EKS) using Helm Chart, it orchestrates a robust development environment within the AWS ecosystem.

Prerequisites

- You have an EKS cluster with AWS Application Load Balancer (ALB) add-on installed. For more information, see [Application load balancing on Amazon Developer Hub](#) and [Installing the AWS Load Balancer Controller add-on](#).
- You have configured a domain name for your Developer Hub instance. The domain name can be a hosted zone entry on Route 53 or managed outside of AWS. For more information, see [Configuring Amazon Route 53 as your DNS service](#) documentation.
- You have an entry in the AWS Certificate Manager (ACM) for your preferred domain name. Make sure to keep a record of your Certificate ARN.
- You have subscribed to **registry.redhat.io**. For more information, see [Red Hat Container Registry Authentication](#).
- You have set the context to the EKS cluster in your current **kubeconfig**. For more information, see [Creating or updating a kubeconfig file for an Amazon EKS cluster](#).
- You have installed **kubect**. For more information, see [Installing or updating kubect](#).
- You have installed Helm 3 or the latest. For more information, see [Using Helm with Amazon EKS](#).

Procedure

1. Go to your terminal and run the following command to add the Helm chart repository containing the Developer Hub chart to your local Helm registry:

```
helm repo add openshift-helm-charts https://charts.openshift.io/
```

2. Create a pull secret using the following command:

```
kubectl create secret docker-registry rhdh-pull-secret \
```

```
--docker-server=registry.redhat.io \
--docker-username=<user_name> \ 1
--docker-password=<password> \ 2
--docker-email=<email> 3
```

- 1 Enter your username in the command.
- 2 Enter your password in the command.
- 3 Enter your email address in the command.

The created pull secret is used to pull the Developer Hub images from the Red Hat Ecosystem.

3. Create a file named **values.yaml** using the following template:

```
global:
  # TODO: Set your application domain name.
  host: <your Developer Hub domain name>

route:
  enabled: false

upstream:
  service:
    # NodePort is required for the ALB to route to the Service
    type: NodePort

ingress:
  enabled: true
  annotations:
    kubernetes.io/ingress.class: alb

    alb.ingress.kubernetes.io/scheme: internet-facing

    # TODO: Using an ALB HTTPS Listener requires a certificate for your own domain. Fill in
    # the ARN of your certificate, e.g.:
    alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:xxx:xxxx:certificate/xxxxxx

    alb.ingress.kubernetes.io/listen-ports: '[{"HTTP": 80}, {"HTTPS":443}]'

    alb.ingress.kubernetes.io/ssl-redirect: '443'

  # TODO: Set your application domain name.
  external-dns.alpha.kubernetes.io/hostname: <your rhdh domain name>

backstage:
```

```

image:
  pullSecrets:
    - rhdh-pull-secret
podSecurityContext:
  # you can assign any random value as fsGroup
  fsGroup: 2000
postgresql:
  image:
    pullSecrets:
      - rhdh-pull-secret
  primary:
    podSecurityContext:
      enabled: true
      # you can assign any random value as fsGroup
      fsGroup: 3000
volumePermissions:
  enabled: true

```

4. Run the following command in your terminal to deploy Developer Hub using the latest version of Helm Chart and using the values.yaml file created in the previous step:

```

helm install rhdh \
  openshift-helm-charts/redhat-developer-hub \
  [--version 1.1.4] \
  --values /path/to/values.yaml

```



NOTE

For the latest chart version, see <https://github.com/openshift-helm-charts/charts/tree/main/charts/redhat/redhat/redhat-developer-hub>

Verification

Wait until the DNS name is responsive, indicating that your Developer Hub instance is ready for use.

3.2. DEPLOYING RED HAT DEVELOPER HUB ON ELASTIC KUBERNETES SERVICE (EKS) USING THE OPERATOR

You can deploy the Developer Hub on EKS using the Red Hat Developer Hub Operator with or without [Operator Lifecycle Manager \(OLM\) framework](#). Following that, you can proceed to install your Developer Hub instance in EKS.

3.2.1. Installing the Red Hat Developer Hub Operator with the OLM framework

Prerequisites

- You have set the context to the EKS cluster in your current **kubeconfig**. For more information, see [Creating or updating a kubeconfig file for an Amazon EKS cluster](#) .
- You have installed **kubect**. For more information, see [Installing or updating kubect](#) .
- You have subscribed to **registry.redhat.io**. For more information, see [Red Hat Container Registry Authentication](#).

- You have installed the Operator Lifecycle Manager (OLM). For more information about installation and troubleshooting, see [How do I get Operator Lifecycle Manager?](#)

Procedure

- Run the following command in your terminal to create the **rhdh-operator** namespace where the Operator is installed:

```
kubectl create namespace rhdh-operator
```

- Create a pull secret using the following command:

```
kubectl -n rhdh-operator create secret docker-registry rhdh-pull-secret \
  --docker-server=registry.redhat.io \
  --docker-username=<user_name> \ 1
  --docker-password=<password> \ 2
  --docker-email=<email> 3
```

- Enter your username in the command.
- Enter your password in the command.
- Enter your email address in the command.

The created pull secret is used to pull the Developer Hub images from the Red Hat Ecosystem.

- Create a **CatalogSource** resource that contains the Operators from the Red Hat Ecosystem:

```
cat <<EOF | kubectl -n rhdh-operator apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: redhat-catalog
spec:
  sourceType: grpc
  image: registry.redhat.io/redhat/redhat-operator-index:v4.15
  secrets:
  - "rhdh-pull-secret"
  displayName: Red Hat Operators
EOF
```

- Create an **OperatorGroup** resource as follows:

```
cat <<EOF | kubectl apply -n rhdh-operator -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: rhdh-operator-group
EOF
```

- Create a **Subscription** resource using the following code:

```
cat <<EOF | kubectl apply -n rhdh-operator -f -
```

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: rhdh
  namespace: rhdh-operator
spec:
  channel: fast
  installPlanApproval: Automatic
  name: rhdh
  source: redhat-catalog
  sourceNamespace: rhdh-operator
  startingCSV: rhdh-operator.v1.1.2
EOF

```

6. Run the following command to verify that the created Operator is running:

```
kubectl -n rhdh-operator get pods -w
```

If the operator pod shows **ImagePullBackOff** status, then you might need permissions to pull the image directly within the Operator deployment's manifest.

TIP

You can include the required secret name in the **deployment.spec.template.spec.imagePullSecrets** list and verify the deployment name using **kubectl get deployment -n rhdh-operator** command:

```

kubectl -n rhdh-operator patch deployment \
  rhdh.fast --patch '{"spec":{"template":{"spec":{"imagePullSecrets":[{"name":"rhdh-pull-secret"}]}}}}' \
  --type=merge

```

7. Update the default configuration of the operator to ensure that Developer Hub resources can start correctly in EKS using the following steps:
 - a. Edit the **backstage-default-config** ConfigMap in the **rhdh-operator** namespace using the following command:

```
kubectl -n rhdh-operator edit configmap backstage-default-config
```

- b. Locate the **db-statefulset.yaml** string and add the **fsGroup** to its **spec.template.spec.securityContext**, as shown in the following example:

```

db-statefulset.yaml: |
  apiVersion: apps/v1
  kind: StatefulSet
  --- TRUNCATED ---
  spec:
  --- TRUNCATED ---
  restartPolicy: Always
  securityContext:
    # You can assign any random value as fsGroup
    fsGroup: 2000

```

```

    serviceAccount: default
    serviceAccountName: default
--- TRUNCATED ---

```

- c. Locate the **deployment.yaml** string and add the **fsGroup** to its specification, as shown in the following example:

```

deployment.yaml: |
  apiVersion: apps/v1
  kind: Deployment
--- TRUNCATED ---
  spec:
    securityContext:
      # You can assign any random value as fsGroup
      fsGroup: 3000
    automountServiceAccountToken: false
--- TRUNCATED ---

```

- d. Locate the **service.yaml** string and change the **type** to **NodePort** as follows:

```

service.yaml: |
  apiVersion: v1
  kind: Service
  spec:
    # NodePort is required for the ALB to route to the Service
    type: NodePort
--- TRUNCATED ---

```

- e. Save and exit.
Wait for a few minutes until the changes are automatically applied to the operator pods.

3.2.2. Installing the Red Hat Developer Hub Operator without the OLM framework

Prerequisites

- You have installed the following commands:
 - **git**
 - **make**
 - **sed**

Procedure

1. Clone the Operator repository to your local machine using the following command:

```
git clone --depth=1 https://github.com/janus-idp/operator.git rhdh-operator && cd rhdh-operator
```

2. Run the following command and generate the deployment manifest:

```
make deployment-manifest
```

The previous command generates a file named **rhhd-operator-<VERSION>.yaml**, which is updated manually.

- Run the following command to apply replacements in the generated deployment manifest:

```
sed -i "s/backstage-operator/rhhd-operator/g" rhhd-operator-*.yaml
sed -i "s/backstage-system/rhhd-operator/g" rhhd-operator-*.yaml
sed -i "s/backstage-controller-manager/rhhd-controller-manager/g" rhhd-operator-*.yaml
```

- Open the generated deployment manifest file in an editor and perform the following steps:
 - Locate the **db-statefulset.yaml** string and add the **fsGroup** to its **spec.template.spec.securityContext**, as shown in the following example:

```
db-statefulset.yaml: |
  apiVersion: apps/v1
  kind: StatefulSet
  --- TRUNCATED ---
  spec:
  --- TRUNCATED ---
  restartPolicy: Always
  securityContext:
    # You can assign any random value as fsGroup
    fsGroup: 2000
  serviceAccount: default
  serviceAccountName: default
  --- TRUNCATED ---
```

- Locate the **deployment.yaml** string and add the **fsGroup** to its specification, as shown in the following example:

```
deployment.yaml: |
  apiVersion: apps/v1
  kind: Deployment
  --- TRUNCATED ---
  spec:
  securityContext:
    # You can assign any random value as fsGroup
    fsGroup: 3000
  automountServiceAccountToken: false
  --- TRUNCATED ---
```

- Locate the **service.yaml** string and change the **type** to **NodePort** as follows:

```
service.yaml: |
  apiVersion: v1
  kind: Service
  spec:
    # NodePort is required for the ALB to route to the Service
    type: NodePort
  --- TRUNCATED ---
```

- Replace the default images with the images that are pulled from the Red Hat Ecosystem:

```
sed -i "s#gcr.io/kubebuilder/kube-rbac-proxy:.*#registry.redhat.io/openshift4/ose-kube-
```

```
rbac-proxy:v4.15#g" rhdh-operator-*.yaml
```

```
sed -i "s#quay.io/janus-idp/operator:.*#registry.redhat.io/rhdh/rhdh-rhel9-operator:1.1#g"
rhdh-operator-*.yaml
```

```
sed -i "s#quay.io/janus-idp/backstage-showcase:.*#registry.redhat.io/rhdh/rhdh-hub-
rhdh-rhel9:1.1#g" rhdh-operator-*.yaml
```

```
sed -i "s#quay.io/fedora/postgresql-15:.*#registry.redhat.io/rhel9/postgresql-15:latest#g"
rhdh-operator-*.yaml
```

5. Add the image pull secret to the manifest in the Deployment resource as follows:

```
--- TRUNCATED ---
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app.kubernetes.io/component: manager
    app.kubernetes.io/created-by: rhdh-operator
    app.kubernetes.io/instance: controller-manager
    app.kubernetes.io/managed-by: kustomize
    app.kubernetes.io/name: deployment
    app.kubernetes.io/part-of: rhdh-operator
    control-plane: controller-manager
  name: rhdh-controller-manager
  namespace: rhdh-operator
spec:
  replicas: 1
  selector:
    matchLabels:
      control-plane: controller-manager
  template:
    metadata:
      annotations:
        kubectrl.kubernetes.io/default-container: manager
      labels:
        control-plane: controller-manager
    spec:
      imagePullSecrets:
        - name: rhdh-pull-secret
--- TRUNCATED ---
```

6. Apply the manifest to deploy the operator using the following command:

```
kubectl apply -f rhdh-operator-VERSION.yaml
```

7. Run the following command to verify that the Operator is running:

```
kubectl -n rhdh-operator get pods -w
```

3.2.3. Installing the Developer Hub instance in EKS

After the Red Hat Developer Hub Operator is installed and running, you can create a Developer Hub instance in EKS.

Prerequisites

- You have an EKS cluster with AWS Application Load Balancer (ALB) add-on installed. For more information, see [Application load balancing on Amazon Elastic Kubernetes Service](#) and [Installing the AWS Load Balancer Controller add-on](#).
- You have configured a domain name for your Developer Hub instance. The domain name can be a hosted zone entry on Route 53 or managed outside of AWS. For more information, see [Configuring Amazon Route 53 as your DNS service](#) documentation.
- You have an entry in the AWS Certificate Manager (ACM) for your preferred domain name. Make sure to keep a record of your Certificate ARN.
- You have subscribed to **registry.redhat.io**. For more information, see [Red Hat Container Registry Authentication](#).
- You have set the context to the EKS cluster in your current **kubecfg**. For more information, see [Creating or updating a kubeconfig file for an Amazon {eks} cluster](#) .
- You have installed **kubect**. For more information, see [Installing or updating kubect](#).

Procedure

1. Create a ConfigMap named **app-config-rhdh** containing the Developer Hub configuration using the following template:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config-rhdh
data:
  "app-config-rhdh.yaml": |
    app:
      title: Red Hat Developer Hub
      baseUrl: https://<rhdh_dns_name>
    backend:
      auth:
        keys:
          - secret: "${BACKEND_SECRET}"
      baseUrl: https://<rhdh_dns_name>
    cors:
      origin: https://<rhdh_dns_name>
```

2. Create a Secret named **secrets-rhdh** and add a key named **BACKEND_SECRET** with a **Base64-encoded** string as value:

```
apiVersion: v1
kind: Secret
metadata:
  name: secrets-rhdh
stringData:
  # TODO: See https://backstage.io/docs/auth/service-to-service-auth/#setup
  BACKEND_SECRET: "xxx"
```



IMPORTANT

Ensure that you use a unique value of **BACKEND_SECRET** for each Developer Hub instance.

You can use the following command to generate a key:

```
node-p'require("crypto").randomBytes(24).toString("base64")'
```

- To enable pulling the PostgreSQL image from the Red Hat Ecosystem Catalog, add the image pull secret in the default service account within the namespace where the Developer Hub instance is being deployed:

```
kubectl patch serviceaccount default \
  -p '{"imagePullSecrets": [{"name": "rhdh-pull-secret"}]}' \
  -n <your_namespace>
```

- Create a Custom Resource file using the following template:

```
apiVersion: rhdh.redhat.com/v1alpha1
kind: Backstage
metadata:
  # TODO: this the name of your Developer Hub instance
  name: my-rhdh
spec:
  application:
    imagePullSecrets:
      - "rhdh-pull-secret"
    route:
      enabled: false
  appConfig:
    configMaps:
      - name: "app-config-rhdh"
  extraEnvs:
    secrets:
      - name: "secrets-rhdh"
```

- Create an Ingress resource using the following template, ensuring to customize the names as needed:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  # TODO: this the name of your Developer Hub Ingress
  name: my-rhdh
annotations:
  alb.ingress.kubernetes.io/scheme: internet-facing

  alb.ingress.kubernetes.io/target-type: ip

  # TODO: Using an ALB HTTPS Listener requires a certificate for your own domain. Fill in
  the ARN of your certificate, e.g.:
  alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:us-xxx:xxxx:certificate/xxxxxx
```

```

alb.ingress.kubernetes.io/listen-ports: '[{"HTTP": 80}, {"HTTPS":443}]'

alb.ingress.kubernetes.io/ssl-redirect: '443'

# TODO: Set your application domain name.
external-dns.alpha.kubernetes.io/hostname: <rhdh_dns_name>

spec:
  ingressClassName: alb
  rules:
    # TODO: Set your application domain name.
    - host: <rhdh_dns_name>
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                # TODO: my-rhdh is the name of your Backstage Custom Resource.
                # Adjust if you changed it!
                name: backstage-my-rhdh
                port:
                  name: http-backend

```

In the previous template, replace ``<rhdh_dns_name>`` with your Developer Hub domain name and update the value of `alb.ingress.kubernetes.io/certificate-arn` with your certificate ARN.

Verification

Wait until the DNS name is responsive, indicating that your Developer Hub instance is ready for use.

3.3. MONITORING AND LOGGING WITH AMAZON WEB SERVICES (AWS) IN RED HAT DEVELOPER HUB

In the Red Hat Developer Hub, monitoring and logging are facilitated through Amazon Web Services (AWS) integration. With features like Amazon CloudWatch for real-time monitoring and Amazon Prometheus for comprehensive logging, you can ensure the reliability, scalability, and compliance of your Developer Hub application hosted on AWS infrastructure.

This integration enables you to oversee, diagnose, and refine your applications in the Red Hat ecosystem, leading to an improved development and operational journey.

3.3.1. Monitoring with Amazon Prometheus

Red Hat Developer Hub provides Prometheus metrics related to the running application. For more information about enabling or deploying Prometheus for EKS clusters, see [Prometheus metrics](#) in the Amazon documentation.

To monitor Developer Hub using [Amazon Prometheus](#), you need to create an Amazon managed service for the Prometheus workspace and configure the ingestion of the Developer Hub Prometheus metrics. For more information, see [Create a workspace](#) and [Ingest Prometheus metrics to the workspace](#) sections in the Amazon documentation.

After ingesting Prometheus metrics into the created workspace, you can configure the metrics scraping to extract data from pods based on specific pod annotations.

3.3.1.1. Configuring annotations for monitoring

You can configure the annotations for monitoring in both Helm deployment and Operator-backed deployment.

Helm deployment

To annotate the backstage pod for monitoring, update your **values.yaml** file as follows:

```
upstream:
  backstage:
    # --- TRUNCATED ---
    podAnnotations:
      prometheus.io/scrape: 'true'
      prometheus.io/path: '/metrics'
      prometheus.io/port: '7007'
      prometheus.io/scheme: 'http'
```

Operator-backed deployment

Procedure

1. As an administrator of the operator, edit the default configuration to add Prometheus annotations as follows:

```
# Update OPERATOR_NS accordingly
OPERATOR_NS=rhdh-operator
kubectl edit configmap backstage-default-config -n "${OPERATOR_NS}"
```

2. Find the **deployment.yaml** key in the ConfigMap and add the annotations to the **spec.template.metadata.annotations** field as follows:

```
deployment.yaml: |-
  apiVersion: apps/v1
  kind: Deployment
  # --- truncated ---
  spec:
    template:
      # --- truncated ---
      metadata:
        labels:
          rhdh.redhat.com/app: # placeholder for 'backstage-<cr-name>'
        # --- truncated ---
        annotations:
          prometheus.io/scrape: 'true'
          prometheus.io/path: '/metrics'
          prometheus.io/port: '7007'
          prometheus.io/scheme: 'http'
      # --- truncated ---
```

3. Save your changes.

Verification

To verify if the scraping works:

1. Use **kubectl** to port-forward the Prometheus console to your local machine as follows:

```
kubectl --namespace=prometheus port-forward deploy/prometheus-server 9090
```

2. Open your web browser and navigate to **http://localhost:9090** to access the Prometheus console.
3. Monitor relevant metrics, such as **process_cpu_user_seconds_total**.

3.3.2. Logging with Amazon CloudWatch logs

Logging within the Red Hat Developer Hub relies on the [winston library](#). By default, logs at the debug level are not recorded. To activate debug logs, you must set the environment variable **LOG_LEVEL** to debug in your Red Hat Developer Hub instance.

3.3.2.1. Configuring the application log level

You can configure the application log level in both Helm deployment and Operator-backed deployment.

Helm deployment

To update the logging level, add the environment variable **LOG_LEVEL** to your Helm chart's **values.yaml** file:

```
upstream:
  backstage:
    # --- Truncated ---
    extraEnvVars:
      - name: LOG_LEVEL
        value: debug
```

Operator-backed deployment

You can modify the logging level by including the environment variable **LOG_LEVEL** in your custom resource as follows:

```
spec:
  # Other fields omitted
  application:
    extraEnvs:
      envs:
        - name: LOG_LEVEL
          value: debug
```

3.3.2.2. Retrieving logs from Amazon CloudWatch

The CloudWatch Container Insights are used to capture logs and metrics for Amazon EKS. For more information, see [Logging for Amazon EKS](#) documentation.

To capture the logs and metrics, [install the Amazon CloudWatch Observability EKS add-on](#) in your cluster. Following the setup of Container Insights, you can access container logs using Logs Insights or Live Tail views.

CloudWatch names the log group where all container logs are consolidated in the following manner:

/aws/containerinsights/<ClusterName>/application

Following is an example query to retrieve logs from the Developer Hub instance:

```
fields @timestamp, @message, kubernetes.container_name
| filter kubernetes.container_name in ["install-dynamic-plugins", "backstage-backend"]
```

3.4. USING AMAZON COGNITO AS AN AUTHENTICATION PROVIDER IN RED HAT DEVELOPER HUB

In this section, Amazon Cognito is an AWS service for adding an authentication layer to Developer Hub. You can sign in directly to the Developer Hub using a user pool or federate through a third-party identity provider.

Although Amazon Cognito is not part of the core authentication providers for the Developer Hub, it can be integrated using the generic OpenID Connect (OIDC) provider.

You can configure your Developer Hub in both Helm Chart and Operator-backed deployments.

Prerequisites

- You have a User Pool or you have created a new one. For more information about user pools, see [Amazon Cognito user pools](#) documentation.



NOTE

Ensure that you have noted the AWS region where the user pool is located and the user pool ID.

- You have created an App Client within your user pool for integrating the hosted UI. For more information, see [Setting up the hosted UI with the Amazon Cognito console](#) .
When setting up the hosted UI using the Amazon Cognito console, ensure to make the following adjustments:
 - In the **Allowed callback URL(s)** section, include the URL **https://<rhdh_url>/api/auth/oidc/handler/frame**. Ensure to replace **<rhdh_url>** with your Developer Hub application's URL, such as, **my.rhdh.example.com**.
 - Similarly, in the **Allowed sign-out URL(s)** section, add **https://<rhdh_url>**. Replace **<rhdh_url>** with your Developer Hub application's URL, such as **my.rhdh.example.com**.
 - Under **OAuth 2.0 grant types**, select **Authorization code grant** to return an authorization code.
 - Under **OpenID Connect scopes**, ensure to select at least the following scopes:
 - OpenID
 - Profile

- Email

Helm deployment

Procedure

1. Edit or create your custom **app-config-rhdh** ConfigMap as follows:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config-rhdh
data:
  "app-config-rhdh.yaml": |
    # --- Truncated ---
    app:
      title: Red Hat Developer Hub

    signInPage: oidc
    auth:
      environment: production
      session:
        secret: ${AUTH_SESSION_SECRET}
      providers:
        oidc:
          production:
            clientId: ${AWS_COGNITO_APP_CLIENT_ID}
            clientSecret: ${AWS_COGNITO_APP_CLIENT_SECRET}
            metadataUrl: ${AWS_COGNITO_APP_METADATA_URL}
            callbackUrl: ${AWS_COGNITO_APP_CALLBACK_URL}
            scope: 'openid profile email'
            prompt: auto
```

2. Edit or create your custom **secrets-rhdh** Secret using the following template:

```
apiVersion: v1
kind: Secret
metadata:
  name: secrets-rhdh
stringData:
  AUTH_SESSION_SECRET: "my super auth session secret - change me!!!"
  AWS_COGNITO_APP_CLIENT_ID: "my-aws-cognito-app-client-id"
  AWS_COGNITO_APP_CLIENT_SECRET: "my-aws-cognito-app-client-secret"
  AWS_COGNITO_APP_METADATA_URL: "https://cognito-idp.
[region].amazonaws.com/[userPoolId]/.well-known/openid-configuration"
  AWS_COGNITO_APP_CALLBACK_URL:
    "https://[rhdh_dns]/api/auth/oidc/handler/frame"
```

3. Add references of both the ConfigMap and Secret resources in your **values.yaml** file:

```
upstream:
  backstage:
    image:
      pullSecrets:
        - rhdh-pull-secret
```

```

podSecurityContext:
  fsGroup: 2000
extraAppConfig:
  - filename: app-config-rhdh.yaml
    configMapRef: app-config-rhdh
extraEnvVarsSecrets:
  - secrets-rhdh

```

4. Upgrade the Helm deployment:

```

helm upgrade rhdh \
  openshift-helm-charts/redhat-developer-hub \
  [--version 1.1.4] \
  --values /path/to/values.yaml

```

Operator-backed deployment

1. Add the following code to your **app-config-rhdh** ConfigMap:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config-rhdh
data:
  "app-config-rhdh.yaml": |
    # --- Truncated ---

    signInPage: oidc
    auth:
      # Production to disable guest user login
      environment: production
      # Providing an auth.session.secret is needed because the oidc provider requires
      session support.
      session:
        secret: ${AUTH_SESSION_SECRET}
      providers:
        oidc:
          production:
            # See https://github.com/backstage/backstage/blob/master/plugins/auth-
            backend-module-oidc-provider/config.d.ts
            clientId: ${AWS_COGNITO_APP_CLIENT_ID}
            clientSecret: ${AWS_COGNITO_APP_CLIENT_SECRET}
            metadataUrl: ${AWS_COGNITO_APP_METADATA_URL}
            callbackUrl: ${AWS_COGNITO_APP_CALLBACK_URL}
            # Minimal set of scopes needed. Feel free to add more if needed.
            scope: 'openid profile email'

            # Note that by default, this provider will use the 'none' prompt which assumes
            that your are already logged on in the IDP.
            # You should set prompt to:
            # - auto: will let the IDP decide if you need to log on or if you can skip login
            when you have an active SSO session
            # - login: will force the IDP to always present a login form to the user
            prompt: auto

```


2. Add the following code to your **secrets-rhdh** Secret:

```

apiVersion: v1
kind: Secret
metadata:
  name: secrets-rhdh
stringData:
  # --- Truncated ---

  # TODO: Change auth session secret.
  AUTH_SESSION_SECRET: "my super auth session secret - change me!!!"

  # TODO: user pool app client ID
  AWS_COGNITO_APP_CLIENT_ID: "my-aws-cognito-app-client-id"

  # TODO: user pool app client Secret
  AWS_COGNITO_APP_CLIENT_SECRET: "my-aws-cognito-app-client-secret"

  # TODO: Replace region and user pool ID
  AWS_COGNITO_APP_METADATA_URL: "https://cognito-idp.
[region].amazonaws.com/[userPoolId]/.well-known/openid-configuration"

  # TODO: Replace <rhdh_dns>
  AWS_COGNITO_APP_CALLBACK_URL:
  "https://[rhdh_dns]/api/auth/oidc/handler/frame"

```

3. Ensure your Custom Resource contains references to both the **app-config-rhdh** ConfigMap and **secrets-rhdh** Secret:

```

apiVersion: rhdh.redhat.com/v1alpha1
kind: Backstage
metadata:
  # TODO: this the name of your Developer Hub instance
  name: my-rhdh
spec:
  application:
    imagePullSecrets:
      - "rhdh-pull-secret"
    route:
      enabled: false
  appConfig:
    configMaps:
      - name: "app-config-rhdh"
  extraEnvs:
    secrets:
      - name: "secrets-rhdh"

```

4. Optional: If you have an existing Developer Hub instance backed by the Custom Resource and you have not edited it, you can manually delete the Developer Hub deployment to recreate it using the operator. Run the following command to delete the Developer Hub deployment:

```
kubectl delete deployment -l app.kubernetes.io/instance=<CR_NAME>
```

Verification

1. Navigate to your Developer Hub web URL and sign in using OIDC authentication, which prompts you to authenticate through the configured AWS Cognito user pool.
2. Once logged in, access **Settings** and verify user details.

CHAPTER 4. RED HAT DEVELOPER HUB INTEGRATION WITH MICROSOFT AZURE KUBERNETES SERVICE (AKS)

You can integrate Developer Hub with Microsoft Azure Kubernetes Service (AKS), which provides a significant advancement in development, offering a streamlined environment for building, deploying, and managing your applications.

This integration requires the deployment of Developer Hub on AKS using one of the following methods:

- The Helm chart
- The Red Hat Developer Hub Operator

4.1. DEPLOYING RED HAT DEVELOPER HUB ON AZURE KUBERNETES SERVICE (AKS)

You can deploy your Developer Hub application on Azure Kubernetes Services (AKS) to access a comprehensive solution for building, testing, and deploying applications.

Prerequisites

- You have an Azure account with active subscription.
- You have installed Azure CLI in your machine and configured the Resource Group and Cluster. For more information, see [How to install the Azure CLI](#).

You can perform the following steps to configure the Resource Group and Cluster:

- To access Azure, ensuring you're logged in to our designated tenant use the following command:

```
az login [--tenant=<optional-directory-name>]
```

- To create a Resource Group, run the following command:

```
az group create --name <your_ResourceGroup> --location <location>
```

TIP

you can retrieve available regions using **az account list-locations -o table**.

- Create an AKS cluster:

```
az aks create \
  --resource-group <your_ResourceGroup> \
  --name <your_ClusterName> \
  --enable-managed-identity \
  --generate-ssh-keys
```

You can refer to **--help** for additional options.

- Connect to your cluster:

```
az aks get-credentials --resource-group <your_ResourceGroup> --name
<your_ClusterName>
```

The previous command configures the Kubernetes client and sets the current context in the **kubeconfig** to point to your AKS cluster.

- You have installed **kubectl**. For more information, see [Installing or updating kubectl](#).
- You have installed Helm 3 or the latest.

Comparison of AKS specifics with the base Developer Hub deployment

- **Permissions issue:** Developer Hub containers might encounter permission-related errors, such as **Permission denied** when attempting certain operations. This error can be addressed by adjusting the **fsGroup** in the **PodSpec.securityContext**.
- **Ingress configuration:** In AKS, configuring ingress is essential for accessing the installed Developer Hub instance. Accessing the Developer Hub instance requires enabling the Routing add-on, an NGINX-based Ingress Controller, using the following commands:

```
az aks approuting enable --resource-group <your_ResourceGroup> --name
<your_ClusterName>
```

TIP

You might need to install the Azure CLI extension **aks-preview**. If the extension is not installed automatically, you might need to install it manually using the following command:

```
az extension add --upgrade -n aks-preview --allow-preview true
```

NOTE

After you install the Ingress Controller, the 'app-routing-system' namespace with the Ingress Controller will be deployed in your cluster. Note the address of your Developer Hub application from the installed Ingress Controller (for example, 108.141.70.228) for later access to the Developer Hub application, later referenced as **<app_address>**.

```
kubectl get svc nginx --namespace app-routing-system -o
jsonpath='{.status.loadBalancer.ingress[0].ip}'
```

- **Namespace management:** You can create a dedicated namespace for Developer Hub deployment in AKS using the following command:

```
kubectl create namespace <your_namespace>
```

4.1.1. Deploying the Red Hat Developer Hub on Azure Kubernetes Service (AKS) using the Helm chart

You can deploy your Developer Hub on AKS using Helm.

Procedure

1. Open terminal and run the following command to add the Helm chart repository:

```
helm repo add openshift-helm-charts https://charts.openshift.io/
```

2. To create **ImagePull Secret**, run the following command:

```
kubectl -n <your_namespace> create secret docker-registry rhdh-pull-secret \
  --docker-server=registry.redhat.io \
  --docker-username=<redhat_user_name> \
  --docker-password=<redhat_password> \
  --docker-email=<email>
```

3. Create a file named **values.yaml** using the following template:

```
global:
  host: <app_address>
route:
  enabled: false
upstream:
  ingress:
    enabled: true
    className: webapprouting.kubernetes.azure.com
    host:
backstage:
  image:
    pullSecrets:
      - rhdh-pull-secret
  podSecurityContext:
    fsGroup: 3000
postgresql:
  image:
    pullSecrets:
      - rhdh-pull-secret
  primary:
    podSecurityContext:
      enabled: true
      fsGroup: 3000
  volumePermissions:
    enabled: true
```

4. To install Helm Chart, run the following command:

```
helm -n <your_namespace> install -f values.yaml <your_deploy_name> openshift-helm-
charts/redhat-developer-hub --version 1.1.1
```

5. Verify the deployment status:

```
kubectl get deploy <your_deploy_name>-developer-hub -n <your_namespace>
```

6. Access the deployed Developer Hub using the URL: https://<app_address>, where <app_address> is the Ingress address obtained earlier (for example, <https://108.141.70.228>).

- To upgrade or delete the deployment, run the following command:

Upgrade command

```
helm -n <your_namespace> upgrade -f values.yaml <your_deploy_name> openshift-helm-charts/redhat-developer-hub --version 1.1.1
```

Delete command

```
helm -n <your_namespace> delete <your_deploy_name>
```

4.1.2. Deploying the Red Hat Developer Hub on Azure Kubernetes Service (AKS) using the Operator

You can deploy your Developer Hub on AKS using the Red Hat Developer Hub Operator.

Procedure

- Obtain the Red Hat Developer Hub Operator manifest file, named **rhdh-operator-*<VERSION>*.yaml**, and modify the default configuration of **db-statefulset.yaml** and **deployment.yaml** by adding the following fragment:

```
securityContext:
  fsGroup: 300
```

Following is the specified locations in the manifests:

```
db-statefulset.yaml: | spec.template.spec
deployment.yaml: | spec.template.spec
```

- Apply the modified Operator manifest to your Kubernetes cluster:

```
kubectl apply -f rhdh-operator-<VERSION>.yaml
```



NOTE

Execution of the previous command is cluster-scoped and requires appropriate cluster privileges.

- Create an **ImagePull Secret** named **rhdh-pull-secret** using your Red Hat credentials to access images from the protected **registry.redhat.io** as shown in the following example:

```
kubectl -n <your_namespace> create secret docker-registry rhdh-pull-secret \
  --docker-server=registry.redhat.io \
  --docker-username=<redhat_user_name> \
  --docker-password=<redhat_password> \
  --docker-email=<email>
```

- Create an Ingress manifest file, named **rhdh-ingress.yaml**, specifying your Developer Hub service name as follows:

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: rhdh-ingress
  namespace: <your_namespace>
spec:
  ingressClassName: webapprouting.kubernetes.azure.com
  rules:
  - http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: backstage-<your-CR-name>
            port:
              name: http-backend

```

- To deploy the created Ingress, run the following command:

```
kubectl -n <your_namespace> apply -f rhdh-ingress.yaml
```

- Create a ConfigMap named **app-config-rhdh** containing the Developer Hub configuration using the following example:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config-rhdh
data:
  "app-config-rhdh.yaml": |
    app:
      title: Red Hat Developer Hub
      baseUrl: https://<app_address>
    backend:
      auth:
        keys:
          - secret: "${BACKEND_SECRET}"
      baseUrl: https://<app_address>
    cors:
      origin: https://<app_address>

```

- Create a Secret named **secrets-rhdh** and add a key named **BACKEND_SECRET** with a **Base64-encoded** string value as shown in the following example:

```

apiVersion: v1
kind: Secret
metadata:
  name: secrets-rhdh
stringData:
  BACKEND_SECRET: "xxx"

```

- Create a Custom Resource (CR) manifest file named **rhdh.yaml** and include the previously created **rhdh-pull-secret** as follows:

```

apiVersion: rhdh.redhat.com/v1alpha1
kind: Backstage
metadata:
  name: <your-rhdh-cr>
spec:
  application:
    imagePullSecrets:
      - rhdh-pull-secret
    appConfig:
      configMaps:
        - name: "app-config-rhdh"
    extraEnvs:
      secrets:
        - name: "secrets-rhdh"

```

9. Apply the CR manifest to your namespace:

```
kubectl -n <your_namespace> apply -f rhdh.yaml
```

10. Access the deployed Developer Hub using the URL: https://<app_address>, where <app_address> is the Ingress address obtained earlier (for example, <https://108.141.70.228>).
11. Optional: To delete the CR, run the following command:

```
kubectl -n <your_namespace> delete -f rhdh.yaml
```

4.2. MONITORING AND LOGGING WITH AZURE KUBERNETES SERVICES (AKS) IN RED HAT DEVELOPER HUB

Monitoring and logging are integral aspects of managing and maintaining Azure Kubernetes Services (AKS) in Red Hat Developer Hub. With features like Managed Prometheus Monitoring and Azure Monitor integration, administrators can efficiently monitor resource utilization, diagnose issues, and ensure the reliability of their containerized workloads.

To enable Managed Prometheus Monitoring, use the **-enable-azure-monitor-metrics** option within either the **az aks create** or **az aks update** command, depending on whether you're creating a new cluster or updating an existing one, such as:

```
az aks create/update --resource-group <your-ResourceGroup> --name <your-Cluster> --enable-azure-monitor-metrics
```

The previous command installs the metrics add-on, which gathers [Prometheus metrics](#). Using the previous command, you can enable monitoring of Azure resources through both native Azure Monitor metrics and Prometheus metrics. You can also view the results in the portal under **Monitoring** → **Insights**. For more information, see [Monitor Azure resources with Azure Monitor](#).

Furthermore, metrics from both the Managed Prometheus service and Azure Monitor can be accessed through Azure Managed Grafana service. For more information, see [Link a Grafana workspace](#) section.

By default, Prometheus uses the minimum ingesting profile, which optimizes ingestion volume and sets default configurations for scrape frequency, targets, and metrics collected. The default settings can be customized through custom configuration. Azure offers various methods, including using different ConfigMaps, to provide scrape configuration and other metric add-on settings. For more information

about default configuration, see [Default Prometheus metrics configuration in Azure Monitor](#) and [Customize scraping of Prometheus metrics in Azure Monitor managed service for Prometheus](#) documentation.

4.2.1. Viewing logs with Azure Kubernetes Services (AKS)

You can access live data logs generated by Kubernetes objects and collect log data in Container Insights within AKS.

Prerequisites

- You have deployed Developer Hub on AKS. For more information, see [Section 4.1, “Deploying Red Hat Developer Hub on Azure Kubernetes Service \(AKS\)”](#).

Procedure

View live logs from your Developer Hub instance

1. Navigate to the Azure Portal.
2. Search for the resource group **<your-ResourceGroup>** and locate your AKS cluster **<your-Cluster>**.
3. Select **Kubernetes resources** → **Workloads** from the menu.
4. Select the **<your-rhdh-cr>-developer-hub** (in case of Helm Chart installation) or **<your-rhdh-cr>-backstage** (in case of Operator-backed installation) deployment.
5. Click **Live Logs** in the left menu.
6. Select the pod.



NOTE

There must be only single pod.

Live log data is collected and displayed.

View real-time log data from the Container Engine

1. Navigate to the Azure Portal.
2. Search for the resource group **<your-ResourceGroup>** and locate your AKS cluster **<your-Cluster>**.
3. Select **Monitoring** → **Insights** from the menu.
4. Go to the **Containers** tab.
5. Find the backend-backstage container and click it to view real-time log data as it's generated by the Container Engine.

4.3. USING MICROSOFT AZURE AS AN AUTHENTICATION PROVIDER IN RED HAT DEVELOPER HUB

The **core-plugin-api** package in Developer Hub comes integrated with Microsoft Azure authentication provider, authenticating signing in using Azure OAuth.

Prerequisites

- You have deployed Developer Hub on AKS. For more information, see [Section 4.1, “Deploying Red Hat Developer Hub on Azure Kubernetes Service \(AKS\)”](#).
- You have created registered your application in Azure portal. For more information, see [Register an application with the Microsoft identity platform](#).

4.3.1. Using Microsoft Azure as an authentication provider in Helm deployment

You can use Microsoft Azure as an authentication provider in Red Hat Developer Hub, when installed using the Helm Chart. For more information, see [Section 4.1.1, “Deploying the Red Hat Developer Hub on Azure Kubernetes Service \(AKS\) using the Helm chart”](#).

Procedure

1. After the application is registered, note down the following:
 - **clientId**: Application (client) ID, found under App **Registration** → **Overview**.
 - **clientSecret**: Secret, found under *App Registration → Certificates & secrets (create new if needed).
 - **tenantId**: Directory (tenant) ID, found under **App Registration** → **Overview**.
2. Ensure the following fragment is included in your Developer Hub ConfigMap:

```
auth:
  environment: production
  providers:
    microsoft:
      production:
        clientId: ${AZURE_CLIENT_ID}
        clientSecret: ${AZURE_CLIENT_SECRET}
        tenantId: ${AZURE_TENANT_ID}
        domainHint: ${AZURE_TENANT_ID}
        additionalScopes:
          - Mail.Send
```

You can either create a new file or add it to an existing one.

3. Apply the ConfigMap to your Kubernetes cluster:

```
kubectl -n <your_namespace> apply -f <app-config>.yaml
```

4. Create or reuse an existing Secret containing Azure credentials and add the following fragment:

```
stringData:
  AZURE_CLIENT_ID: <value-of-clientId>
```

```
AZURE_CLIENT_SECRET: <value-of-clientSecret>
AZURE_TENANT_ID: <value-of-tenantId>
```

5. Apply the secret to your Kubernetes cluster:

```
kubectl -n <your_namespace> apply -f <azure-secrets>.yaml
```

6. Ensure your **values.yaml** file references the previously created ConfigMap and Secret:

```
upstream:
  backstage:
    ...
    extraAppConfig:
      - filename: ...
        configMapRef: <app-config-containing-azure>
    extraEnvVarsSecrets:
      - <secret-containing-azure>
```

7. Optional: If the Helm Chart is already installed, upgrade it:

```
helm -n <your_namespace> upgrade -f <your-values.yaml> <your_deploy_name> redhat-developer/backstage --version 1.1.4
```

8. Optional: If your **rhdh.yaml** file is not changed, for example, you only updated the ConfigMap and Secret referenced from it, refresh your Developer Hub deployment by removing the corresponding pods:

```
kubectl -n <your_namespace> delete pods -l backstage.io/app=backstage-<your-rhdh-cr>
```

4.3.2. Using Microsoft Azure as an authentication provider in Operator-backed deployment

You can use Microsoft Azure as an authentication provider in Red Hat Developer Hub, when installed using the Operator. For more information, see [Section 2.2, “Deploying Red Hat Developer Hub on OpenShift Container Platform using the Operator”](#).

Procedure

1. After the application is registered, note down the following:
 - **clientId**: Application (client) ID, found under App **Registration** → **Overview**.
 - **clientSecret**: Secret, found under *App Registration → Certificates & secrets (create new if needed).
 - **tenantId**: Directory (tenant) ID, found under **App Registration** → **Overview**.
2. Ensure the following fragment is included in your Developer Hub ConfigMap:

```
auth:
  environment: production
  providers:
    microsoft:
      production:
```

```

clientId: ${AZURE_CLIENT_ID}
clientSecret: ${AZURE_CLIENT_SECRET}
tenantId: ${AZURE_TENANT_ID}
domainHint: ${AZURE_TENANT_ID}
additionalScopes:
  - Mail.Send

```

You can either create a new file or add it to an existing one.

3. Apply the ConfigMap to your Kubernetes cluster:

```
kubectl -n <your_namespace> apply -f <app-config>.yaml
```

4. Create or reuse an existing Secret containing Azure credentials and add the following fragment:

```

stringData:
  AZURE_CLIENT_ID: <value-of-clientId>
  AZURE_CLIENT_SECRET: <value-of-clientSecret>
  AZURE_TENANT_ID: <value-of-tenantId>

```

5. Apply the secret to your Kubernetes cluster:

```
kubectl -n <your_namespace> apply -f <azure-secrets>.yaml
```

6. Ensure your Custom Resource manifest contains references to the previously created ConfigMap and Secret:

```

apiVersion: rhdh.redhat.com/v1alpha1
kind: Backstage
metadata:
  name: <your-rhdh-cr>
spec:
  application:
    imagePullSecrets:
      - rhdh-pull-secret
    route:
      enabled: false
  appConfig:
    configMaps:
      - name: <app-config-containing-azure>
  extraEnvs:
    secrets:
      - name: <secret-containing-azure>

```

7. Apply your Custom Resource manifest:

```
kubectl -n <your_namespace> apply -f rhdh.yaml
```

8. Optional: If your **rhdh.yaml** file is not changed, for example, you only updated the ConfigMap and Secret referenced from it, refresh your Developer Hub deployment by removing the corresponding pods:

```
kubectl -n <your_namespace> delete pods -l backstage.io/app=backstage-<your-rhdh-cr>
```

CHAPTER 5. ROLE-BASED ACCESS CONTROL (RBAC) IN RED HAT DEVELOPER HUB

Role-Based Access Control is a security paradigm that restricts access to authorized users. This feature includes defining roles with specific permissions and then assigning those roles to the users.

The Red Hat Developer Hub uses RBAC to improve the permission system within the platform. The RBAC feature in Developer Hub introduces an administrator role and leverages the organizational structure including teams, groups, and users by facilitating efficient access control.

5.1. PERMISSION POLICIES CONFIGURATION

There are two approaches to configure the permission policies in Red Hat Developer Hub, including:

- Configuration of permission policies administrators
- Configuration of permission policies defined in an external file

5.1.1. Configuration of permission policies administrators

The permission policies for users and groups in the Developer Hub are managed by permission policy administrators. Only permission policy administrators can access the Role-Based Access Control REST API.

The purpose of configuring policy administrators is to enable a specific, restricted number of authenticated users to access the RBAC REST API. The permission policies are defined in a **policy.csv** file, which is referenced in the **app-config-rhdh** ConfigMap. OpenShift platform administrators or cluster administrators can perform this task with access to the namespace where Red Hat Developer Hub is deployed.

You can set the credentials of a permission policy administrator in the **app-config.yaml** file as follows:

```
permission:
  enabled: true
rbac:
  admin:
    users:
      - name: user:default/joeuser
```

5.1.2. Configuration of permission policies defined in an external file

You can follow this approach of configuring the permission policies before starting the Red Hat Developer Hub. If permission policies are defined in an external file, then you can import the same file in the Developer Hub. The permission policies need to be defined in Casbin rules format. For information about the Casbin rules format, see [Basics of Casbin rules](#).

The following is an example of permission policies configuration:

```
p, role:default/guests, catalog-entity, read, deny
```

```
p, role:default/guests, catalog.entity.create, create, deny
```

```
g, user:default/<USER_TO_ROLE>, role:default/guests
```

If a defined permission does not contain an action associated with it, then add **use** as a policy. See the following example:

p, role:default/guests, kubernetes.proxy, use, deny

You can define the **policy.csv** file path in the **app-config.yaml** file:

```
permission:
  enabled: true
  rbac:
    policies-csv-file: /some/path/rbac-policy.csv
```

5.1.2.1. Mounting policy.csv file to the Developer Hub Helm Chart

When the Red Hat Developer Hub is deployed with the Helm Chart, then you must define the **policy.csv** file by mounting it to the Developer Hub Helm Chart.

You can add your **policy.csv** file to the Developer Hub Helm Chart by creating a **configMap** and mounting it.

Prerequisites

- You are logged in to your OpenShift Container Platform account using the OpenShift Container Platform web console.
- Red Hat Developer Hub is installed and deployed using Helm Chart.
For more information about installing the Red Hat Developer Hub on OpenShift Container Platform using Helm Chart, see [Section 2.1, "Deploying Red Hat Developer Hub on OpenShift Container Platform using Helm Chart"](#).

Procedure

1. In OpenShift Container Platform, create a ConfigMap to hold the policies as shown in the following example:

Example ConfigMap

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: rbac-policy
  namespace: rhdh
data:
  rbac-policy.csv: |
    p, role:default/guests, catalog-entity, read, allow
    p, role:default/guests, catalog.entity.create, create, allow

    g, user:default/<YOUR_USER>, role:default/guests
```

2. In the Developer Hub Helm Chart, go to **Root Schema** → **Backstage chart schema** → **Backstage parameters** → **Backstage container additional volume mounts**.
3. Select **Add Backstage container additional volume mounts** and add the following values:
 - **mountPath: opt/app-root/src/rbac**

- Name: **rbac-policy**
4. Add the RBAC policy to the **Backstage container additional volumes** in the Developer Hub Helm Chart:
 - name: **rbac-policy**
 - configMap
 - **defaultMode: 420**
 - **name: rbac-policy**
 5. Update the policy path in the **app-config.yaml** file as follows:

Example **app-config.yaml** file

```

permission:
  enabled: true
rbac:
  policies-csv-file: ./rbac/rbac-policy.csv

```

5.1.3. Permission policies in Red Hat Developer Hub

Permission policies in Red Hat Developer Hub are a set of rules to govern access to resources or functionalities. These policies state the authorization level that is granted to users based on their roles. The permission policies are implemented to maintain security and confidentiality within a given environment.

The following permission policies are supported in the Developer Hub:

Catalog permissions

Name	Resource type	Policy	Description
catalog.entity.read	catalog-entity	read	Allows user or role to read from the catalog
catalog.entity.create		create	Allows user or role to create catalog entities, including registering an existing component in the catalog
catalog.entity.refresh	catalog-entity	update	Allows user or role to refresh a single or multiple entities from the catalog
catalog.entity.delete	catalog-entity	delete	Allows user or role to delete a single or multiple entities from the catalog
catalog.location.read		read	Allows user or role to read a single or multiple locations from the catalog

Name	Resource type	Policy	Description
catalog.location.create		create	Allows user or role to create locations within the catalog
catalog.location.delete		delete	Allows user or role to delete locations from the catalog

Scaffolder permissions

Name	Resource type	Policy	Description
scaffolder.action.execute	scaffolder-action		Allows the execution of an action from a template
scaffolder.template.parameter.read	scaffolder-template	read	Allows user or role to read a single or multiple one parameters from a template
scaffolder.template.step.read	scaffolder-template	read	Allows user or role to read a single or multiple steps from a template

RBAC permissions

Name	Resource type	Policy	Description
policy.entity.read	policy-entity	read	Allows user or role to read permission policies and roles
policy.entity.create	policy-entity	create	Allows user or role to create a single or multiple permission policies and roles
policy.entity.update	policy-entity	update	Allows user or role to update a single or multiple permission policies and roles
policy.entity.delete	policy-entity	delete	Allows user or role to delete a single or multiple permission policies and roles

Kubernetes permissions

Name	Resource type	Policy	Description
kubernetes.proxy			Allows user or role to access the proxy endpoint

5.2. MANAGING ROLE-BASED ACCESS CONTROLS (RBAC) USING THE RED HAT DEVELOPER HUB WEB UI

Administrators can use the Developer Hub web interface (Web UI) to allocate specific roles and permissions to individual users or groups. Allocating roles ensures that access to resources and functionalities is regulated across the Developer Hub.

With the administrator role in Developer Hub, you can assign permissions to users and groups, which allow users or groups to view, create, modify, and delete the roles using the Developer Hub Web UI.

To access the RBAC features in the Web UI, you must install and configure the **@janus-idp/backstage-plugin-rbac** plugin as a dynamic plugin. For more information about installing a dynamic plugin, see [Chapter 6, *Dynamic plugin installation*](#).

After you install the **@janus-idp/backstage-plugin-rbac** plugin, the **Administration** option appears at the bottom of the sidebar. When you can click **Administration**, the RBAC tab appears by default, displaying all of the existing roles created in the Developer Hub. In the RBAC tab, you can also view the total number of users, groups, and the total number of permission policies associated with a role. You can also edit or delete a role using the **Actions** column.

5.2.1. Creating a role in the Red Hat Developer Hub Web UI

You can create a role in the Red Hat Developer Hub using the Web UI.

Prerequisites

- You have an administrator role in the Developer Hub.
- You have installed the **@janus-idp/backstage-plugin-rbac** plugin in Developer Hub. For more information, see [Chapter 6, *Dynamic plugin installation*](#).
- You have configured the required permission policies. For more information, see [Section 5.1, *“Permission policies configuration”*](#).

Procedure

1. Go to **Administration** at the bottom of the sidebar in the Developer Hub. The **RBAC** tab appears, displaying all the created roles in the Developer Hub.
2. (Optional) Click any role to view the role information on the **OVERVIEW** page.
3. Click **CREATE** to create a role.
4. Enter the name and description of the role in the given fields and click **NEXT**.
5. Add users and groups using the search field, and click **NEXT**.
6. Select **Plugin** and **Permission** from the drop-downs in the **Add permission policies** section.

7. Select or clear the **Policy** that you want to set in the **Add permission policies** section, and click **NEXT**.
8. Review the added information in the **Review and create** section.
9. Click **CREATE**.

Verification

The created role appears in the list available in the **RBAC** tab.

5.2.2. Editing a role in the Red Hat Developer Hub Web UI

You can edit a role in the Red Hat Developer Hub using the Web UI.



NOTE

The policies generated from a **policy.csv** or ConfigMap file cannot be edited or deleted using the Developer Hub Web UI.

Prerequisites

- You have an administrator role in the Developer Hub.
- You have installed the **@janus-idp/backstage-plugin-rbac** plugin in Developer Hub. For more information, see [Chapter 6, Dynamic plugin installation](#).
- You have configured the required permission policies. For more information, see [Section 5.1, "Permission policies configuration"](#).
- The role that you want to edit is created in the Developer Hub.

Procedure

1. Go to **Administration** at the bottom of the sidebar in the Developer Hub. The **RBAC** tab appears, displaying all the created roles in the Developer Hub.
2. (Optional) Click any role to view the role information on the **OVERVIEW** page.
3. Select the edit icon for the role that you want to edit.
4. Edit the details of the role, such as name, description, users and groups, and permission policies, and click **NEXT**.
5. Review the edited details of the role and click **SAVE**.

After editing a role, you can view the edited details of a role on the **OVERVIEW** page of a role. You can also edit a role's users and groups or permissions by using the edit icon on the respective cards on the **OVERVIEW** page.

5.2.3. Deleting a role in the Red Hat Developer Hub Web UI

You can delete a role in the Red Hat Developer Hub using the Web UI.



NOTE

The policies generated from a **policy.csv** or ConfigMap file cannot be edited or deleted using the Developer Hub Web UI.

Prerequisites

- You have an administrator role in the Developer Hub.
- You have installed the **@janus-idp/backstage-plugin-rbac** plugin in Developer Hub. For more information, see [Chapter 6, Dynamic plugin installation](#).
- You have configured the required permission policies. For more information, see [Section 5.1, "Permission policies configuration"](#).
- The role that you want to delete is created in the Developer Hub.

Procedure

1. Go to **Administration** at the bottom of the sidebar in the Developer Hub. The **RBAC** tab appears, displaying all the created roles in the Developer Hub.
2. (Optional) Click any role to view the role information on the **OVERVIEW** page.
3. Select the delete icon from the **Actions** column for the role that you want to delete. **Delete this role?** pop-up appears on the screen.
4. Click **DELETE**.

5.3. ROLE-BASED ACCESS CONTROL (RBAC) REST API

Red Hat Developer Hub provides RBAC REST API that you can use to manage the permissions and roles in the Developer Hub. This API supports you to facilitate and automate the maintenance of Developer Hub permission policies and roles.

Using the RBAC REST API, you can perform the following actions:

- Retrieve information about all permission policies or specific permission policies, or roles
- Create, update, or delete a permission policy or a role
- Retrieve permission policy information about static plugins

The RBAC REST API requires the following components:

Authorization

The RBAC REST API requires Bearer token authorization for the permitted user role. For development purposes, you can access a web console in a browser. When you refresh a token request in the list of network requests, you find the token in the response JSON.

Authorization: Bearer \$token

For example, on the **Homepage** of the Developer Hub, you can navigate to the **Network** tab and search for the **query?term=** network call. Alternatively, you can go to the **Catalog** page and select any network call with **entity-facets** to acquire the Bearer token.

HTTP methods

The RBAC REST API supports the following HTTP methods for API requests:

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

Base URL

The base URL for RBAC REST API requests is **http://SERVER:PORT/api/permission/policies**, such as **http://localhost:7007/api/permission/policies**.

Endpoints

RBAC REST API endpoints, such as **/api/permission/policies/[kind]/[namespace]/[name]** for specified kind, namespace, and username, are the URI that you append to the base URL to access the corresponding resource.

Example request URL for **/api/permission/policies/[kind]/[namespace]/[name]** endpoint is:

http://localhost:7007/api/permission/policies/user/default/johndoe



NOTE

If at least one permission is assigned to **user:default/johndoe**, then the example request URL mentioned previously returns a result if sent in a **GET** response with a valid authorization token. However, if permission is only assigned to roles, then the example request URL does not return an output.

Request data

HTTP **POST** requests in the RBAC REST API may require a JSON request body with data to accompany the request.

Example **POST** request URL and JSON request body data for **http://localhost:7007/api/permission/policies**:

```
{
  "entityReference": "role:default/test",
  "permission": "catalog-entity",
  "policy": "delete",
  "effect": "allow"
}
```

HTTP status codes

The RBAC REST API supports the following HTTP status codes to return as responses:

- **200 OK**: The request was successful.
- **201 Created**: The request resulted in a new resource being successfully created.

- **204** No Content: The request was successful, but there is no additional content to send in the response payload.
- **400** Bad Request: input error with the request
- **401** Unauthorized: lacks valid authentication for the requested resource
- **403** Forbidden: refusal to authorize request
- **404** Not Found: could not find requested resource
- **409** Conflict: request conflict with the current state and the target resource

5.3.1. Sending requests with the RBAC REST API using a REST client or curl utility

The RBAC REST API enables you to interact with the permission policies and roles in Developer Hub without using the user interface. You can send RBAC REST API requests using any REST client or curl utility.

Prerequisites

- Red Hat Developer Hub is installed and running. For more information about installing Red Hat Developer Hub, see [Section 2.1, “Deploying Red Hat Developer Hub on OpenShift Container Platform using Helm Chart”](#).
- You have access to the Developer Hub.

Procedure

1. Identify a relevant API endpoint to which you want to send a request, such as **POST /api/permission/policies**. Adjust any request details according to your use case.

For REST client:

- Authorization: Enter the generated token from the web console.
- HTTP method: Set to **POST**.
- URL: Enter the RBAC REST API base URL and endpoint such as **http://localhost:7007/api/permission/policies**.

For curl utility:

- **-X**: Set to **POST**
- **-H**: Set the following header:
Content-type: application/json
Authorization: Bearer \$token

\$token is the requested token from the web console in a browser.

- URL: Enter the following RBAC REST API base URL endpoint, such as **http://localhost:7007/api/permission/policies**
- **-d**: Add a request JSON body

Example request:

```
curl -X POST "http://localhost:7007/api/permission/policies" -d
'{"entityReference":"role:default/test", "permission": "catalog-entity", "policy": "read",
"effect":"allow"}' -H "Content-Type: application/json" -H "Authorization: Bearer $token" -v
```

2. Execute the request and review the response.

5.3.2. Supported RBAC REST API endpoints

The RBAC REST API provides the following endpoints for managing permission policies and roles in the Developer Hub and for retrieving information about the policies and roles.

5.3.2.1. Permission policies

The RBAC REST API supports the following endpoints for managing permission policies in the Red Hat Developer Hub.

[GET] /api/permission/policies

Returns permission policies list for all users.

Example response (JSON)

```
[
  {
    "entityReference": "role:default/test",
    "permission": "catalog-entity",
    "policy": "read",
    "effect": "allow"
  },
  {
    "entityReference": "role:default/test",
    "permission": "catalog.entity.create",
    "policy": "use",
    "effect": "allow"
  }
]
```

[GET] /api/permission/policies/{kind}/{namespace}/{name}

Returns permission policies related to the specified entity reference.

Table 5.1. Request parameters

Name	Description	Type	Requirement
kind	Kind of the entity	String	Required
namespace	Namespace of the entity	String	Required
name	Username related to the entity	String	Required

Example response (JSON)

```
[
  {
    "entityReference": "role:default/test",
    "permission": "catalog-entity",
    "policy": "read",
    "effect": "allow"
  },
  {
    "entityReference": "role:default/test",
    "permission": "catalog.entity.create",
    "policy": "use",
    "effect": "allow"
  }
]
```

[POST] /api/permission/policies

Creates a permission policy for a specified entity.

Table 5.2. Request parameters

Name	Description	Type	Requirement
entityReference	Reference values of an entity including namespace and name	String	Required
permission	Type of the permission	String	Required
policy	Read or write policy to the permission	String	Required
effect	Indication of allowing or not allowing the policy	String	Required

Example request body (JSON)

```
{
  "entityReference": "role:default/test",
  "permission": "catalog-entity",
  "policy": "read",
  "effect": "allow"
}
```

Example response

201 Created

[PUT] /api/permission/policies/{kind}/{namespace}/{name}

Updates a permission policy for a specified entity.

Request parameters

The request body contains the **oldPolicy** and **newPolicy** objects:

Name	Description	Type	Requirement
permission	Type of the permission	String	Required
policy	Read or write policy to the permission	String	Required
effect	Indication of allowing or not allowing the policy	String	Required

Example request body (JSON)

```
{
  "oldPolicy": {
    "permission": "catalog-entity",
    "policy": "read",
    "effect": "deny"
  },
  "newPolicy": {
    "permission": "policy-entity",
    "policy": "read",
    "effect": "allow"
  }
}
```

Example response

```
200
```

[DELETE] /api/permission/policies/{kind}/{namespace}/{name}?permission={value1}&policy={value2}&effect={value3}

Deletes a permission policy added to the specified entity.

Table 5.3. Request parameters

Name	Description	Type	Requirement
kind	Kind of the entity	String	Required
namespace	Namespace of the entity	String	Required
name	Username related to the entity	String	Required
permission	Type of the permission	String	Required

Name	Description	Type	Requirement
policy	Read or write policy to the permission	String	Required
effect	Indication of allowing or not allowing the policy	String	Required

Example response

204 No Content

[GET] /api/permission/plugins/policies

Returns permission policies for all static plugins.

Example response (JSON)

```
[
  {
    "pluginId": "catalog",
    "policies": [
      {
        "permission": "catalog-entity",
        "policy": "read"
      },
      {
        "permission": "catalog.entity.create",
        "policy": "create"
      },
      {
        "permission": "catalog-entity",
        "policy": "delete"
      },
      {
        "permission": "catalog-entity",
        "policy": "update"
      },
      {
        "permission": "catalog.location.read",
        "policy": "read"
      },
      {
        "permission": "catalog.location.create",
        "policy": "create"
      },
      {
        "permission": "catalog.location.delete",
        "policy": "delete"
      }
    ]
  },
  ...
]
```

5.3.2.2. Roles

The RBAC REST API supports the following endpoints for managing roles in the Red Hat Developer Hub.

[GET] /api/permission/roles

Returns all roles in Developer Hub.

Example response (JSON)

```
[
  {
    "memberReferences": ["user:default/pataknight"],
    "name": "role:default/guests"
  },
  {
    "memberReferences": [
      "group:default/janus-authors",
      "user:default/matt"
    ],
    "name": "role:default/rbac_admin"
  }
]
```

[GET] /api/permission/roles/{kind}/{namespace}/{name}

Creates a role in Developer Hub.

Table 5.4. Request parameters

Name	Description	Type	Requirement
body	The memberReferences , group , namespace , and name the new role to be created.	Request body	Required

Example request body (JSON)

```
{
  "memberReferences": ["group:default/test"],
  "name": "role:default/test_admin"
}
```

Example response

```
201 Created
```

[PUT] /api/permission/roles/{kind}/{namespace}/{name}

Updates **memberReferences**, **kind**, **namespace**, or **name** for a role in Developer Hub.

Request parameters

The request body contains the **oldRole** and **newRole** objects:

Name	Description	Type	Requirement
body	The memberReferences , group , namespace , and name the new role to be created.	Request body	Required

Example request body (JSON)

```
{
  "oldRole": {
    "memberReferences": ["group:default/test"],
    "name": "role:default/test_admin"
  },
  "newRole": {
    "memberReferences": ["group:default/test", "user:default/test2"],
    "name": "role:default/test_admin"
  }
}
```

Example response

```
200 OK
```

[DELETE] /api/permission/roles/{kind}/{namespace}/{name}?memberReferences=<VALUE>

Deletes the specified user or group from a role in Developer Hub.

Table 5.5. Request parameters

Name	Description	Type	Requirement
kind	Kind of the entity	String	Required
namespace	Namespace of the entity	String	Required
name	Username related to the entity	String	Required
memberReferences	Associated group information	String	Required

Example response

```
204
```

[DELETE] /api/permission/roles/{kind}/{namespace}/{name}

Deletes a specified role from Developer Hub.

Table 5.6. Request parameters

Name	Description	Type	Requirement
kind	Kind of the entity	String	Required
namespace	Namespace of the entity	String	Required
name	Username related to the entity	String	Required

Example response

204

CHAPTER 6. DYNAMIC PLUGIN INSTALLATION

The dynamic plugin support is based on the backend plugin manager package, which is a service that scans a configured root directory (**dynamicPlugins.rootDirectory** in the app config) for dynamic plugin packages and loads them dynamically.

You can use the dynamic plugins that come preinstalled with Red Hat Developer Hub or install external dynamic plugins from a public NPM registry.

6.1. VIEWING INSTALLED PLUGINS

Using the Dynamic Plugins Info front-end plugin, you can view plugins that are currently installed in your Red Hat Developer Hub application. This plugin is enabled by default.

Procedure

1. Open your Developer Hub application and click **Administration**.
2. Go to the **Plugins** tab to view a list of installed plugins and related information.

6.2. PREINSTALLED DYNAMIC PLUGINS

Red Hat Developer Hub is preinstalled with a selection of dynamic plugins. The dynamic plugins that require custom configuration are disabled by default.

For a complete list of dynamic plugins that are preinstalled in this release of Developer Hub, see the [Dynamic plugins support matrix](#).

Upon application startup, for each plugin that is disabled by default, the **install-dynamic-plugins init container** within the Developer Hub pod log displays a message similar to the following:

```
===== Skipping disabled dynamic plugin ./dynamic-plugins/dist/backstage-plugin-catalog-backend-
module-github-dynamic
```

To enable this plugin, add a package with the same name to the Helm chart and change the value in the **disabled** field to 'false'. For example:

```
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: ./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-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.

6.2.1. Preinstalled dynamic plugin descriptions and details



IMPORTANT

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.

There are 56 plugins available in Red Hat Developer Hub. See the following table for more information:

Table 6.1. Dynamic plugins support matrix

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
3scale	Backend	@janus-idp/backstage-plugin-3scale-backend	The 3scale Backstage provider plugin synchronizes the 3scale content into the Backstage catalog.	1.4.7	Red Hat Tech Preview	./dynamic-plugins/dist/janus-idp-backstage-plugin-3scale-backend-dynamic	THREE_SCALE_BASE_URL THREE_SCALE_ACCESS_TOKEN	Disabled
AAP	Backend	@janus-idp/backstage-plugin-aap-backend		1.5.5	Red Hat Tech Preview	./dynamic-plugins/dist/janus-idp-backstage-plugin-aap-backend-dynamic	AAP_BASE_URL AAP_AUTH_TOKEN	Disabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
ACR	Frontend	@janus-idp/backstage-plugin-acr		1.2.28	Red Hat Tech Preview	./dynamic-plugins/dist/janus-idp-backstage-plugin-acr		Disabled
Analytics Provider Segment	Frontend	@janus-idp/backstage-plugin-analytics-provider-segment	This plugin provides an implementation of the Backstage Analytics API for Segment. Once installed and configured, analytics events will be sent to Segment as your users navigate and use your Backstage instance.	1.2.11	Production	./dynamic-plugins/dist/janus-idp-backstage-plugin-analytics-provider-segment	SEGMENT_WRITE_KEY SEGMENT_MASK_IP SEGMENT_TEST_MODE	Disabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Argo CD	Frontend	@roadiehq/backstage-plugin-argo-cd	Backstage plugin to view and interact with Argo CD.	2.4.1	Production	./dynamic-plugins/dist/roadiehq-backstage-plugin-argo-cd		Disabled
Argo CD	Backend	@roadiehq/backstage-plugin-argo-cd-backend	Backstage plugin Argo CD backend	2.14.5	Production	./dynamic-plugins/dist/roadiehq-backstage-plugin-argo-cd-backend-dynamic	ARGO_CD_USERNAME ARGO_CD_PASSWORD ARGO_CD_INSTANCE_URL ARGO_CD_AUTH_TOKEN ARGO_CD_INSTANCE_URL2 ARGO_CD_AUTH_TOKEN2	Disabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Argo CD	Backend	@roadiehq/scaffolder-backend-argocd		1.1.23	Community Support	./dynamic-plugins/dist/roadiehq-scaffolder-backend-argocd-dynamic	ARGO_CD_USERNAME ARGO_CD_PASSWORD ARGO_CD_INSTANCE1_URL ARGO_CD_AUTH_TOKEN ARGO_CD_INSTANCE2_URL ARGO_CD_AUTH_TOKEN2	Disabled
Azure Devops	Frontend	@backstage/plugin-azure-devops		0.3.12	Community Support	./dynamic-plugins/dist/backstage-plugin-azure-devops		Disabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Azure Devops	Backend	@backstage/plugin-azure-devops-backend	Azure DevOps backend plugin that contains the API for retrieving builds, pull requests, etc. which is used by the Azure DevOps frontend plugin.	0.5.5	Community Support	./dynamic-plugins/dist/backendstage-plugin-azure-devops-backend-dynamic	AZURE_TOKEN AZURE_ORG	Disabled
Azure Devops	Backend	@backstage/plugin-scaffolder-backend-module-azure	The azure module for @backstage/plugin-scaffolder-backend	0.1.5	Community Support	./dynamic-plugins/dist/backendstage-plugin-scaffolder-backend-module-azure-dynamic		Enabled
Bitbucket	Backend	@backstage/plugin-catalog-backend-module-bitbucket-cloud	A Backstage catalog backend module that helps integrate towards Bitbucket Cloud.	0.1.28	Community Support	./dynamic-plugins/dist/backendstage-plugin-catalog-backend-module-bitbucket-cloud-dynamic	BITBUCKET_WORKSPACE	Disabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Bitbucket	Backend	@backstage/plugin-catalog-backend-module-bitbucket-server	A Backstage catalog backend module that helps integrate towards Bitbucket Server.	0.1.26	Community Support	./dynamic-plugins/dist/backend-module-bitbucket-server-dynamic	BITBUCKET_HOST	Disabled
Bitbucket	Backend	@backstage/plugin-scaffolder-backend-module-bitbucket-cloud	The Bitbucket Cloud module for @backstage/plugin-scaffolder-backend	0.1.3	Community Support	./dynamic-plugins/dist/backend-module-bitbucket-cloud-dynamic		Enabled
Bitbucket	Backend	@backstage/plugin-scaffolder-backend-module-bitbucket-server	The Bitbucket Server module for @backstage/plugin-scaffolder-backend.	0.1.3	Community Support	./dynamic-plugins/dist/backend-module-bitbucket-server-dynamic		Enabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Datadog	Frontend	@roadiehq/backstage-plugin-datadog	Embed Datadog graphs and dashboards into Backstage.	2.2.6	Community Support	./dynamic-plugins/dist/roadiehq-backstage-plugin-datadog		Disabled
Dynatrace	Frontend	@backstage/plugin-dynatrace	A Backstage plugin that integrates towards Dynatrace.	9.0.0	Community Support	./dynamic-plugins/dist/backstage-plugin-dynatrace		Disabled
Dynamic Plugins	Frontend	@janus-idp/backstage-plugin-dynamic-plugins-info	Dynamic Plugins Info plugin for Backstage.	1.0.2	Production	@janus-idp/backstage-plugin-dynamic-plugins-info		Enabled
Gerrit	Backend	@backstage/plugin-scaffolder-backend-module-gerrit	The gerrit module for @backstage/plugin-scaffolder-backend.	0.1.5	Community Support	./dynamic-plugins/dist/backstage-plugin-scaffolder-backend-module-gerrit-dynamic		Enabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Github	Backend	@backstage/plugin-catalog-backend-module-github	A Backstage catalog backend module that helps integrate towards Github	0.5.3	Community Support	./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-dynamic	GITHUB_ORG	Disabled
Github	Backend	@backstage/plugin-catalog-backend-module-github-org	The github-org backend module for the catalog plugin.	0.1.0	Community Support	./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-org-dynamic	GITHUB_URL GITHUB_ORG	Disabled
Github	Frontend	@backstage/plugin-github-actions	A Backstage plugin that integrates towards GitHub Actions	0.6.11	Community Support	./dynamic-plugins/dist/backstage-plugin-github-actions		Disabled
Github	Frontend	@backstage/plugin-github-issues	A Backstage plugin that integrates towards GitHub Issues	0.2.19	Community Support	./dynamic-plugins/dist/backstage-plugin-github-issues		Disabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Github	Backend	@backstage/plugin-scaffolder-backend-module-github	The github module for @backstage/plugin-scaffolder-backend.	0.2.3	Community Support	./dynamic-plugins/dist/backstage-plugin-scaffolder-backend-module-github-dynamic		Enabled
Github	Frontend	@roadiehq/backstage-plugin-github-insights	Backstage plugin to provide Readmes, Top Contributors and other widgets.	2.3.27	Community Support	./dynamic-plugins/dist/roadiehq-backstage-plugin-github-insights		Disabled
Github	Frontend	@roadiehq/backstage-plugin-github-pull-requests	Backstage plugin to view and interact with GitHub pull requests .	2.5.24	Community Support	./dynamic-plugins/dist/roadiehq-backstage-plugin-github-pull-requests		Disabled
Github	Frontend	@roadiehq/backstage-plugin-security-insights	Backstage plugin to add security insights for GitHub repos.	2.3.15	Community Support	./dynamic-plugins/dist/roadiehq-backstage-plugin-security-insights		Disabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Gitlab	Backend	@backstage/plugin-catalog-backend-module-gitlab	Extracts repositories out of an GitLab instance.	0.3.10	Community Support	./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-gitlab-dynamic		Disabled
Gitlab	Backend	@backstage/plugin-scaffolder-backend-module-gitlab	A module for the scaffolder backend that lets you interact with gitlab	0.2.16	Community Support	./dynamic-plugins/dist/backstage-plugin-scaffolder-backend-module-gitlab-dynamic		Disabled
Gitlab	Frontend	@immobiliarelabs/backstage-plugin-gitlab	Backstage plugin to interact with GitLab	6.4.0	Community Support	./dynamic-plugins/dist/immobiliarelabs-backstage-plugin-gitlab		Disabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Gitlab	Backend	@immobiliarelabs/backstage-plugin-gitlab-backend	Backstage plugin to interact with GitLab	6.4.0	Community Support	./dynamic-plugins/dist/immobiliarelabs-backstage-plugin-gitlab-backend-dynamic	GITLAB_HOST GITLAB_TOKEN	Disabled
Jenkins	Frontend	@backstage/plugin-jenkins	A Backstage plugin that integrates towards Jenkins	0.9.5	Community Support	./dynamic-plugins/dist/backstage-plugin-jenkins		Disabled
Jenkins	Backend	@backstage/plugin-jenkins-backend	A Backstage backend plugin that integrates towards Jenkins	0.3.7	Community Support	./dynamic-plugins/dist/backstage-plugin-jenkins-backend-dynamic	JENKINS_URL JENKINS_USERNAME JENKINS_TOKEN	Disabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Jfrog Artifactory	Frontend	@janus-idp/backstage-plugin-jfrog-artifactory	The Jfrog Artifactory plugin displays information about your container images within the Jfrog Artifactory registry.	1.2.28	Red Hat Tech Preview	./dynamic-plugins/dist/janus-idp-backstage-plugin-jfrog-artifactory	ARTIFACTORY_URL ARTIFACTORY_TOKEN ARTIFACTORY_SECURE	Disabled
Jira	Frontend	@roadiehq/backstage-plugin-jira	Backstage plugin to view and interact with Jira	2.5.4	Community Support	./dynamic-plugins/dist/roadiehq-backstage-plugin-jira		Disabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Keycloak	Backend	@janus-idp/backend-stage-plugin-keycloak-backend	The Keycloak backend plugin integrates Keycloak into Backstage.	1.8.6	Production	./dynamic-plugins/dist/janus-idp-backend-stage-plugin-keycloak-backend-dynamic	KEYCLOAK_BACKEND_URL KEYCLOAK_LOGIN_REALM KEYCLOAK_REALM KEYCLOAK_CLIENT_ID KEYCLOAK_CLIENT_SECRET	Disabled
Kubernetes	Frontend	@backstage/plugin-kubernetes	A Backstage plugin that integrates towards Kubernetes	0.11.5	Community Support	./dynamic-plugins/dist/backstage-plugin-kubernetes		Enabled
Kubernetes	Backend	@backstage/plugin-kubernetes-backend	A Backstage backend plugin that integrates towards Kubernetes	0.15.3	Production	./dynamic-plugins/dist/backstage-plugin-kubernetes-backend-dynamic	K8S_CLUSTER_NAME K8S_CLUSTER_URL K8S_CLUSTER_TOKEN	Enabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Kubernetes	Frontend	@janus-idp/backstage-plugin-topology	The Topology plugin enables you to visualize the workloads such as Deployment, Job, Daemon set, Stateful set, CronJob, and Pods powering any service on the Kubernetes cluster.	1.18.8	Production	./dynamic-plugins/dist/janus-idp-backstage-plugin-topology		Enabled
Lighthouse	Frontend	@backstage/plugin-lighthouse	A Backstage plugin that integrates towards Lighthouse	0.4.15	Community Support	./dynamic-plugins/dist/backstage-plugin-lighthouse		Disabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Nexus Repository Manager	Frontend	@janus-idp/backstage-plugin-nexus-repository-manager	The Nexus Repository Manager plugin displays the information about your build artifacts that are available in the Nexus Repository Manager in your Backstage application.	1.4.28	Red Hat Tech Preview	./dynamic-plugins/dist/janus-idp-backstage-plugin-nexus-repository-manager		Disabled
OCM	Frontend	@janus-idp/backstage-plugin-ocm	The Open Cluster Management (OCM) plugin integrates your Backstage instance with the MultiClusterHub and MultiCluster engines of OCM.	3.7.5	Production	./dynamic-plugins/dist/janus-idp-backstage-plugin-ocm		Disabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
OCM	Backend	@janus-idp/backstage-plugin-ocm-backend		3.5.7	Production	./dynamic-plugins/dist/janus-idp-backstage-plugin-ocm-backend-dynamic	OCM_HUB_NAME OCM_HUB_URL moc_infra_token	Disabled
Pagerduty	Frontend	@pagerduty/backstage-plugin	A Backstage plugin that integrates towards PagerDuty	0.9.3	Community Support	./dynamic-plugins/dist/pagerduty-backstage-plugin		Disabled
Quay	Frontend	@janus-idp/backstage-plugin-quay	The Quay plugin displays the information about your container images within the Quay registry in your Backstage application.	1.5.10	Production	./dynamic-plugins/dist/janus-idp-backstage-plugin-quay		Disabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Quay	Backend	@janus-idp/backstage-scaffolder-backend-module-quay	This module provides Backstage template actions for Quay.	1.3.5	Production	./dynamic-plugins/dist/janus-idp-backstage-scaffolder-backend-module-quay-dynamic		Enabled
RBAC	Frontend	@janus-idp/backstage-plugin-rbac	RBAC frontend plugin for Backstage.	1.15.5	Production	./dynamic-plugins/dist/janus-idp-backstage-plugin-rbac		Disabled
Regex	Backend	@janus-idp/backstage-scaffolder-backend-module-regex	This plugin provides Backstage template actions for RegExp.	1.3.5	Production	./dynamic-plugins/dist/janus-idp-backstage-scaffolder-backend-module-regex-dynamic		Enabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Scaffolder	Backend	@roadiehq/scaffolder-backend-module-utils	This contains a collection of actions to use in scaffolder templates.	1.13.6	Community Support	./dynamic-plugins/dist/roadiehq-scaffolder-backend-module-utils-dynamic		Enabled
Service Now	Backend	@janus-idp/backstage-scaffolder-backend-module-servicenow	This plugin provides Backstage template actions for Service Now.	1.3.5	Red Hat Tech Preview	./dynamic-plugins/dist/janus-idp-backstage-scaffolder-backend-module-servicenow-dynamic	SERVICE_NOW_BASE_URL SERVICE_NOW_USERNAME SERVICE_NOW_PASSWORD	Disabled
SonarQuibe	Frontend	@backstage/plugin-sonarquibe	A Backstage plugin to display SonarQuibe code quality and security results.	0.7.12	Community Support	./dynamic-plugins/dist/backstage-plugin-sonarquibe		Disabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
SonarQu be	Backend	@backstage/plugin-sonarqu be-backend		0.2.15	Community Support	./dynamic-plugins/dist/backstage-plugin-sonarqu be-backend-dynamic	SONARQUBE_URL SONARQUBE_TOKEN	Disabled
SonarQu be	Backend	@janus-idp/backstage-scaffolder-backend-module-sonarqu be	This module provides Backstage template actions for SonarQu be.	1.3.5	Red Hat Tech Preview	./dynamic-plugins/dist/janus-idp-backstage-scaffolder-backend-module-sonarqu be-dynamic		Disabled
Tech Radar	Frontend	@backstage/plugin-tech-radar	A Backstage plugin that lets you display a Tech Radar for your organization	0.6.13	Community Support	./dynamic-plugins/dist/backstage-plugin-tech-radar		Disabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Techdocs	Frontend	@backstage/plugin-techdocs	The Backstage plugin that renders technical documentation for your components	1.10.0	Production	./dynamic-plugins/dist/backstage-plugin-techdocs		Enabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Techdocs	Backend	@backstage/plugin-techdocs-backend	The Backstage backend plugin that renders technical documentation for your components	1.9.6	Production	./dynamic-plugins/dist/backend-plugin-techdocs-backend-dynamic	TECHDOCS_BACKEND_PLUGIN_TYPE TECHDOCS_GENERATOR_TYPE TECHDOCS_PUBLISHER_TYPE BUCKET_NAME BUCKET_REGION_VAULT BUCKET_URL AWS_ACCESS_KEY_ID AWS_SECRET_ACCESS_KEY	Enabled

Name	Role	Plugin	Description	Version	Support Level	Path	Required Variables	Default
Tekton	Frontend	@janus-idp/backstage-plugin-tekton	The Tekton plugin enables you to visualize the Pipeline Run resources available on the Kubernetes cluster.	3.5.12	Production	./dynamic-plugins/dist/janus-idp-backstage-plugin-tekton		Disabled

6.3. INSTALLATION OF DYNAMIC PLUGINS USING THE HELM CHART

You can deploy a Developer Hub instance using a Helm chart, which is a flexible installation method. With the Helm chart, you can sideload dynamic plugins into your Developer Hub instance without having to recompile your code or rebuild the container.

To install dynamic plugins in Developer Hub using Helm, add the following **global.dynamic** parameters in your Helm chart:

- **plugins:** the dynamic plugins list intended for installation. By default, the list is empty. You can populate the plugins list with the following fields:
 - **package:** a package specification for the dynamic plugin package that you want to install. You can use a package for either a local or an external dynamic plugin installation. For a local installation, use a path to the local folder containing the dynamic plugin. For an external installation, use a package specification from a public NPM repository.
 - **integrity** (required for external packages): an integrity checksum in the form of **<alg>-<digest>** specific to the package. Supported algorithms include **sha256**, **sha384** and **sha512**.
 - **pluginConfig:** an optional plugin-specific **app-config** YAML fragment. See plugin configuration for more information.
 - **disabled:** disables the dynamic plugin if set to **true**. Default: **false**.
- **includes:** a list of YAML files utilizing the same syntax.



NOTE

The **plugins** list in the **includes** file is merged with the **plugins** list in the main Helm values. If a plugin package is mentioned in both **plugins** lists, the **plugins** fields in the main Helm values override the **plugins** fields in the **includes** file. The default configuration includes the **dynamic-plugins.default.yaml** file, which contains all of the dynamic plugins preinstalled in Developer Hub, whether enabled or disabled by default.

6.3.1. Obtaining the integrity checksum

To obtain the integrity checksum, enter the following command:

```
npm view <package name>@<version> dist.integrity
```

6.3.2. Example Helm chart configurations for dynamic plugin installations

The following examples demonstrate how to configure the Helm chart for specific types of dynamic plugin installations.

Configuring a local plugin and an external plugin when the external plugin requires a specific app-config

```
global:
  dynamic:
    plugins:
      - package: <alocal package-spec used by npm pack>
      - package: <external package-spec used by npm pack>
      integrity: sha512-<some hash>
      pluginConfig: ...
```

Disabling a plugin from an included file

```
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: <some imported plugins listed in dynamic-plugins.default.yaml>
      disabled: true
```

Enabling a plugin from an included file

```
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: <some imported plugins listed in dynamic-plugins.custom.yaml>
      disabled: false
```

Enabling a plugin that is disabled in an included file

```
global:
```

```

dynamic:
  includes:
    - dynamic-plugins.default.yaml
  plugins:
    - package: <some imported plugins listed in dynamic-plugins.custom.yaml>
  disabled: false

```

6.3.3. Installing external dynamic plugins using a Helm chart

The NPM registry contains external dynamic plugins that you can use for demonstration purposes. For example, the following community plugins are available in the **janus-idp** organization in the NPMJS repository:

- Notifications (frontend and backend)
- Kubernetes actions (scaffolder actions)

To install the Notifications and Kubernetes actions plugins, include them in the Helm chart values in the **global.dynamic.plugins** list as shown in the following example:

```

global:
  dynamic:
    plugins:
      - package: '@janus-idp/plugin-notifications-backend-dynamic@1.3.6'
        # Integrity can be found at https://registry.npmjs.org/@janus-idp/plugin-notifications-backend-dynamic
        integrity: 'sha512-
Qd8pniy1yRx+x7LnwjzQ6k9zP+C1yex24MaCcx7dGDPT/XbTokwoSZr4baSSn8jUA6P45NUUevu1d629
mG4JGQ=='
      - package: '@janus-idp/plugin-notifications@1.1.12'
        # https://registry.npmjs.org/@janus-idp/plugin-notifications

        integrity: 'sha512-
GCdEuHRQek3ay428C8C4wWgxjNpNwCXgldFbUUFGCLLkBFSSaOEw+XaBvWaBGtQ5BLgE3jQEUx
a+422uzSYC5oQ=='
      pluginConfig:
        dynamicPlugins:
          frontend:
            janus-idp.backstage-plugin-notifications:
              applcons:
                - name: notificationsIcon
                  module: NotificationsPlugin
                  importName: NotificationsActiveIcon
              dynamicRoutes:
                - path: /notifications
                  importName: NotificationsPage
                  module: NotificationsPlugin
              menuItem:
                icon: notificationsIcon
                text: Notifications
              config:
                pollingIntervalMs: 5000
      - package: '@janus-idp/backstage-scaffolder-backend-module-kubernetes-dynamic@1.3.5'
        # https://registry.npmjs.org/@janus-idp/backstage-scaffolder-backend-module-kubernetes-

```

```

dynamic
  integrity: 'sha512-
19ie+FM3QHxWYPyYzE0uNdl5K8M4vGZ0SPeeTw85XPROY1DrIY7rMm2G0XT85L0ZmntHVwc9qW
+SbHolPg/qRA=='
  proxy:
    endpoints:
      /explore-backend-completed:
        target: 'http://localhost:7017'
  - package: '@dfatwork-pkgs/search-backend-module-explore-wrapped-dynamic@0.1.3-next.1'
    # https://registry.npmjs.org/@dfatwork-pkgs/search-backend-module-explore-wrapped-dynamic
    integrity: 'sha512-
mv6LS8UOve+eumoMCVypGcd7b/L36IH2z11tGKVrt+m65VzQI4FgAJr9kNCrjUZPMyh36KVGIljYqsu9+
kgzH5A=='
  - package: '@dfatwork-pkgs/plugin-catalog-backend-module-test-dynamic@0.0.0'
    # https://registry.npmjs.org/@dfatwork-pkgs/plugin-catalog-backend-module-test-dynamic
    integrity: 'sha512-
YsrZMThxJk7cYJU9FtAcsTCx9ICChpytK254TfGb3iMAYQyVcZnr5AA/AU+hezFnXLsr6gj8PP7z/mCZie
uuDA=='

```

6.4. INSTALLING EXTERNAL PLUGINS IN AN AIR-GAPPED ENVIRONMENT

You can install external plugins in an air-gapped environment by setting up a custom NPM registry. To configure the NPM registry URL and authentication information for dynamic plugin packages, see [Using a custom NPM registry for dynamic plugin packages](#).

6.5. USING A CUSTOM NPM REGISTRY FOR DYNAMIC PLUGIN PACKAGES

You can configure the NPM registry URL and authentication information for dynamic plugin packages using a Helm chart. For dynamic plugin packages obtained through **npm pack**, you can use a **.npmrc** file.

Using the Helm chart, add the **.npmrc** file to the NPM registry by creating a secret named **dynamic-plugins-npmrc** with the following content:

```

apiVersion: v1
kind: Secret
metadata:
  name: dynamic-plugins-npmrc
type: Opaque
stringData:
  .npmrc: |
    registry=<registry-url>
    //<registry-url>:_authToken=<auth-token>
  ...

```

6.6. BASIC CONFIGURATION OF DYNAMIC PLUGINS

Some dynamic plugins require environment variables to be set. If a mandatory environment variable is not set, and the plugin is enabled, then the application might fail at startup.

The mandatory environment variables for each plugin are listed in the [Dynamic plugins support matrix](#).



NOTE

Zib-bomb detection When installing some dynamic plugin containing large files, if the installation script considers the package archive to be a Zib-Bomb, the installation fails.

To increase the maximum permitted size of a file inside a package archive, you can increase the **MAX_ENTRY_SIZE** environment value of the deployment **install-dynamic-plugins initContainer** from the default size of **20000000** bytes.

6.7. INSTALLATION AND CONFIGURATION OF ANSIBLE AUTOMATION PLATFORM

The Ansible Automation Platform (AAP) plugin synchronizes the accessible templates including job templates and workflow job templates from AAP into your Developer Hub catalog.



IMPORTANT

The Ansible Automation Platform 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.

6.7.1. For administrators

6.7.1.1. Installing and configuring the AAP Backend plugin

The AAP backend plugin allows you to configure one or multiple providers using your **app-config.yaml** configuration file in Developer Hub.

Prerequisites

- Your Developer Hub application is installed and running.
- You have created an account in Ansible Automation Platform.

Installation

The AAP backend 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/janus-idp-backstage-plugin-aap-backend-dynamic`
- disabled: `false`

Basic configuration

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

- **AAP_BASE_URL**: Base URL of the service
- **AAP_AUTH_TOKEN**: Authentication token for the service

Advanced configuration

1. You can use the **aap** marker to configure the **app-config.yaml** file of Developer Hub as follows:

```
catalog:
  providers:
    aap:
      dev:
        baseUrl: ${AAP_BASE_URL}
        authorization: 'Bearer ${AAP_AUTH_TOKEN}'
        owner: <owner>
        system: <system>
        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 }
```

6.7.1.2. Log lines for AAP Backend plugin troubleshoot

When you start your Developer Hub application, you can see the following log lines:

```
[1] 2023-02-13T15:26:09.356Z catalog info Discovered ResourceEntity API type=plugin
target=AapResourceEntityProvider:dev
[1] 2023-02-13T15:26:09.423Z catalog info Discovered ResourceEntity Red Hat Event (DEV, v1.2.0)
type=plugin target=AapResourceEntityProvider:dev
[1] 2023-02-13T15:26:09.620Z catalog info Discovered ResourceEntity Red Hat Event (TEST, v1.1.1)
type=plugin target=AapResourceEntityProvider:dev
[1] 2023-02-13T15:26:09.819Z catalog info Discovered ResourceEntity Red Hat Event (PROD,
v1.1.1) type=plugin target=AapResourceEntityProvider:dev
[1] 2023-02-13T15:26:09.819Z catalog info Applying the mutation with 3 entities type=plugin
target=AapResourceEntityProvider:dev
```

6.7.2. For users

6.7.2.1. Accessing templates from AAP in Developer Hub

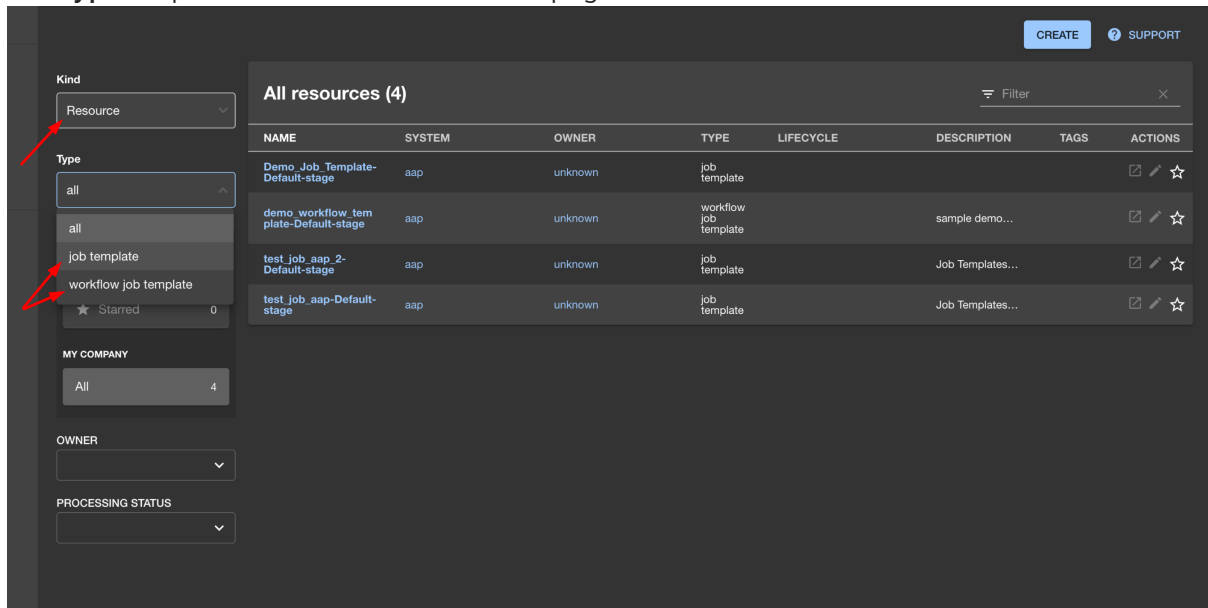
When you have configured the AAP backend plugin successfully, it synchronizes the templates including job templates and workflow job templates from AAP and displays them on the Developer Hub Catalog page as Resources.

Prerequisites

- Your Developer Hub application is installed and running.
- You have installed the AAP backend plugin. For installation and configuration instructions, see [Section 6.7.1.1, “Installing and configuring the AAP Backend plugin”](#) .

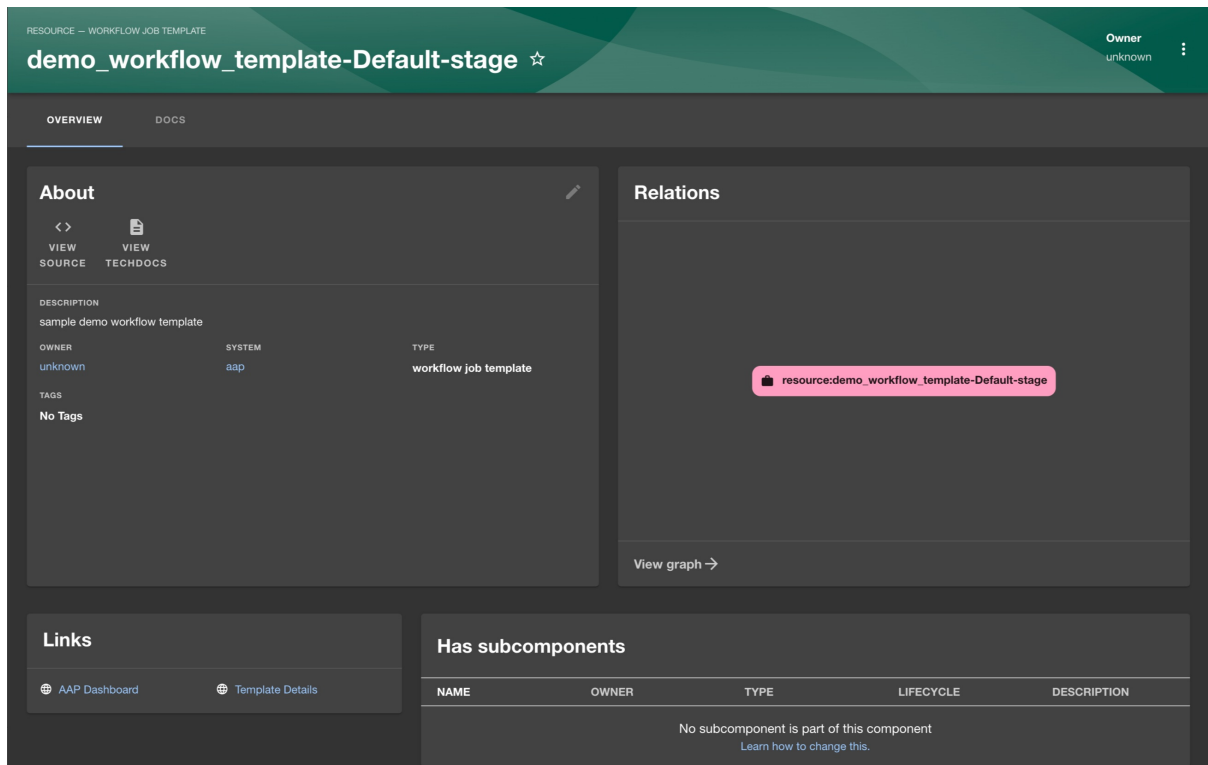
Procedure

1. Open your Developer Hub application and Go to the **Catalog** page.
2. Select **Resource** from the **Kind** drop-down and **job template** or **workflow job template** from the **Type** drop-down on the left side of the page.



A list of all the available templates from AAP appears on the page.

3. Select a template from the list.
The **OVERVIEW** tab appears containing different cards, such as:
 - **About:** Provides detailed information about the template.
 - **Relations:** Displays the visual representation of the template and associated aspects.
 - **Links:** Contains links to the AAP dashboard and the details page of the template.
 - **Has subcomponents:** Displays a list of associated subcomponents.



6.8. INSTALLATION AND CONFIGURATION OF 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.

6.8.1. For administrators

6.8.1.1. 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/janus-idp-backstage-plugin-keycloak-backend-dynamic
        disabled: false
```

6.8.1.2. Basic configuration

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

- **KEYCLOAK_BASE_URL**
- **KEYCLOAK_LOGIN_REALM**

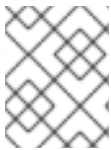
- **KEYCLOAK_REALM**
- **KEYCLOAK_CLIENT_ID**
- **KEYCLOAK_CLIENT_SECRET**

6.8.1.3. 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 . Note that the newer versions of Keycloak omit the /auth context path.	""	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
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

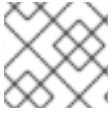
When using client credentials, the access type must be set to **confidential** and service accounts must be enabled. You must also add the following roles from the **realm-management** client role:

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

6.8.1.4. 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.

6.8.2. For users**6.8.2.1. Import of users and groups in Developer Hub using the Keycloak plugin**

After configuring the plugin successfully, the plugin imports the users and groups each time when started.

**NOTE**

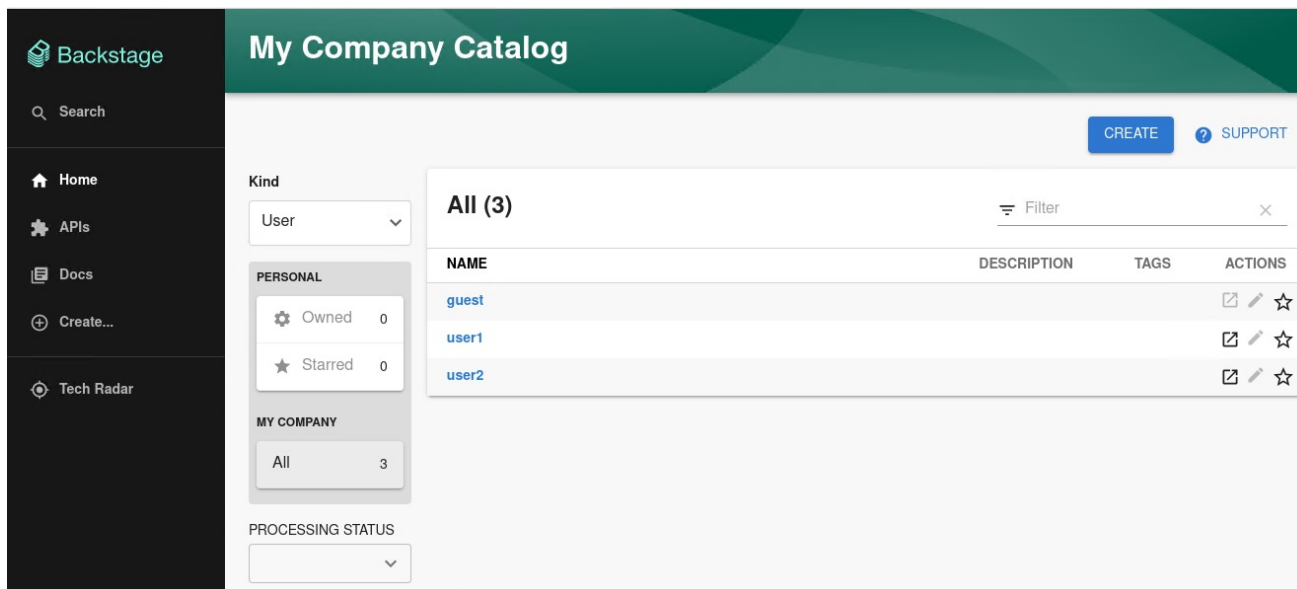
If you set up a schedule, users and groups will also be imported.

After the first import is complete, you can select **User** to list the users from the catalog page:

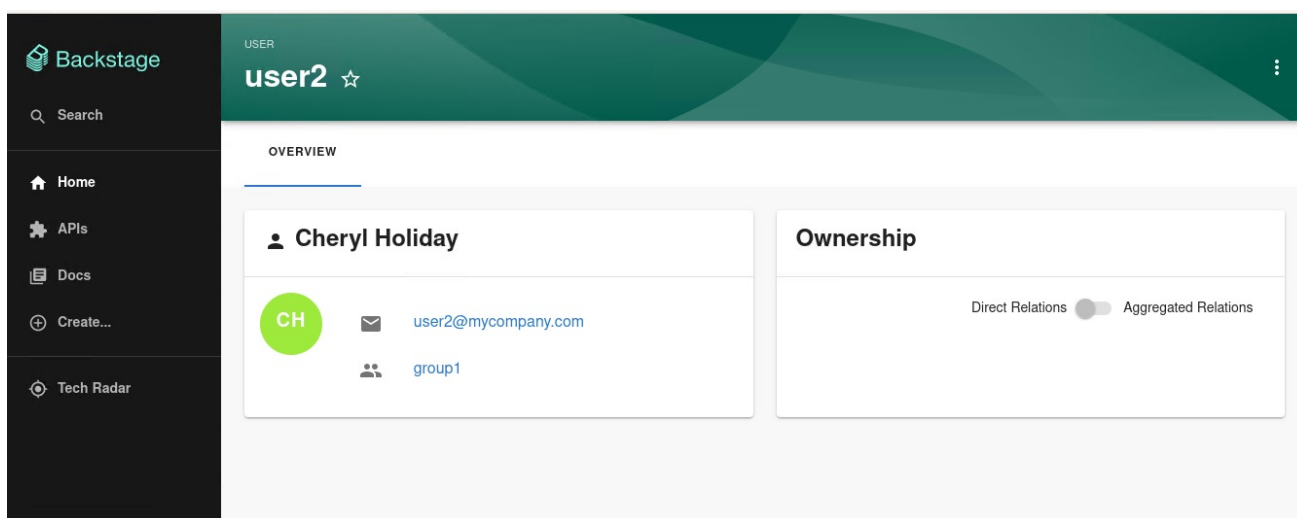
The screenshot shows the Backstage interface for 'My Company Catalog'. The left sidebar contains navigation options: Home, APIs, Docs, Create..., and Tech Radar. The main content area displays a 'Kind' filter dropdown menu with options: Component, API, Component, Group, Location, System, Template, and User. The 'User' option is selected. Below the dropdown, a summary shows 'All 1'. The main table displays a list of users with the following columns: NAME, SYSTEM, OWNER, TYPE, LIFECYCLE, DESCRIPTION, TAGS, and ACTION. The table contains one entry:

NAME	SYSTEM	OWNER	TYPE	LIFECYCLE	DESCRIPTION	TAGS	ACTION
example-website	examples	guests	website	experimental			🔗 ✎

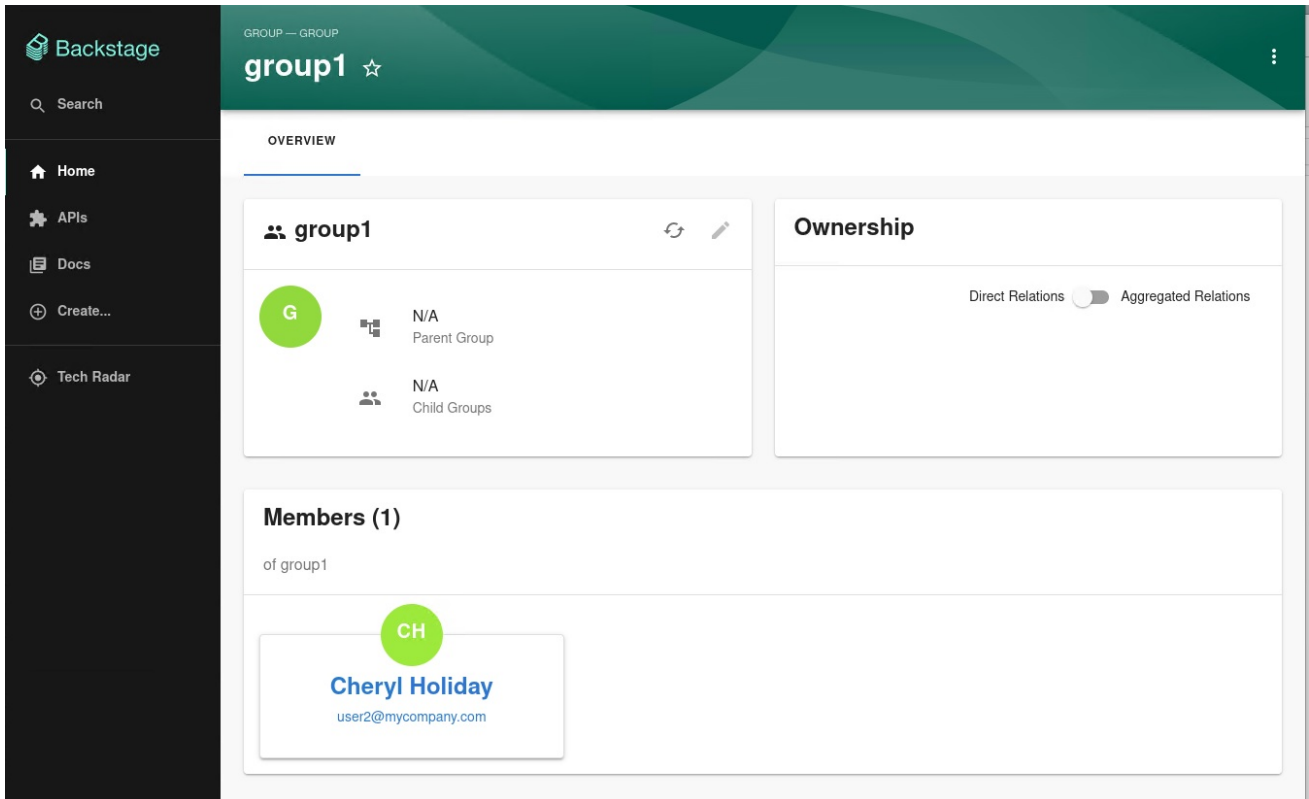
You can see the list of users on the page:



When you select a user, you can see the information imported from Keycloak:



You can also select a group, view the list, and select or view the information imported from Keycloak for a group:



6.9. INSTALLATION AND CONFIGURATION OF NEXUS REPOSITORY MANAGER

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.

6.9.1. For administrators

6.9.1.1. Installing and configuring the Nexus Repository Manager plugin

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/janus-idp-backstage-plugin-nexus-repository-manager
      disabled: false

```

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>`,

```

For additional information about installing and configuring dynamic plugins, see the [Chapter 6, Dynamic plugin installation](#) section.

6.9.2. For users

6.9.2.1. Using the Nexus Repository Manager plugin in Developer Hub

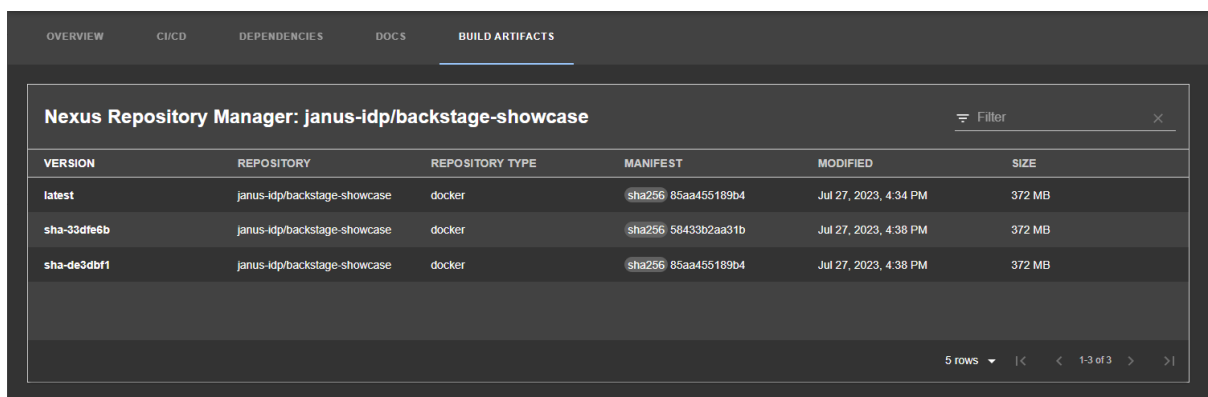
The Nexus Repository Manager is a front-end plugin that enables you to view the information about build artifacts.

Prerequisites

- Your Developer Hub application is installed and running.
- You have installed the Nexus Repository Manager plugin. For the installation process, see [Section 6.9.1.1, “Installing and configuring the Nexus Repository Manager plugin”](#).

Procedure

1. Open your Developer Hub application and select a component from the **Catalog** page.
2. Go to the **BUILD ARTIFACTS** tab.
The **BUILD ARTIFACTS** tab contains a list of build artifacts and related information, such as **VERSION**, **REPOSITORY**, **REPOSITORY TYPE**, **MANIFEST**, **MODIFIED**, and **SIZE**.



VERSION	REPOSITORY	REPOSITORY TYPE	MANIFEST	MODIFIED	SIZE
latest	janus-idp/backstage-showcase	docker	sha256 85aa455189b4	Jul 27, 2023, 4:34 PM	372 MB
sha-33dfe6b	janus-idp/backstage-showcase	docker	sha256 58433b2aa31b	Jul 27, 2023, 4:38 PM	372 MB
sha-de3dbf1	janus-idp/backstage-showcase	docker	sha256 85aa455189b4	Jul 27, 2023, 4:38 PM	372 MB

6.10. INSTALLATION AND CONFIGURATION OF TEKTON

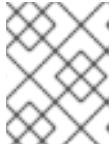
You can use the Tekton plugin to visualize the results of CI/CD pipeline runs on your Kubernetes or OpenShift clusters. The plugin allows users to visually see high level status of all associated tasks in the pipeline for their applications.

6.10.1. For administrators

6.10.1.1. Installation

Prerequisites

- You have installed and configured the **@backstage/plugin-kubernetes** and **@backstage/plugin-kubernetes-backend** dynamic plugins. For more information about installing dynamic plugins, see [Chapter 6, *Dynamic plugin installation*](#).
- 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'

```

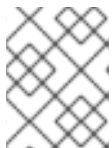
- Add the following label to the resources so that the Kubernetes plugin gets the Kubernetes resources from the requested entity:

```

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/janus-idp-backstage-plugin-tekton
      disabled: false

```

6.10.2. For users

6.10.2.1. Using the Tekton plugin in RHDH

You can use the Tekton front-end plugin to view **PipelineRun** resources.














Prerequisites

- You have installed the Red Hat Developer Hub (RHDH).
- You have installed the Tekton plugin. For the installation process, see [Installing and configuring the Tekton plugin](#).

Procedure














1. Open your RHDH application and select a component from the **Catalog** page.
2. Go to the **CI** tab.

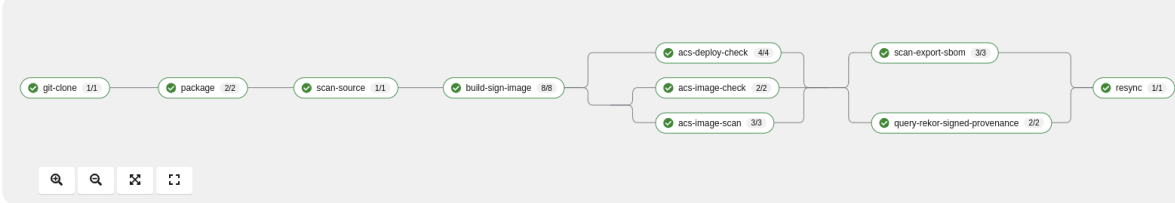
The **CI** tab displays the list of PipelineRun resources associated with a Kubernetes cluster. The list contains pipeline run details, such as **NAME**, **VULNERABILITIES**, **STATUS**, **TASK STATUS**, **STARTED**, and **DURATION**.

Pipeline Runs							Search
NAME	VULNERABILITIES	STATUS	TASK STATUS	STARTED	DURATION	ACTIONS	
> PLR 4ac96b4d-6cfa-4154-b2bc-27e27b4df977	-	✔ Succeeded	<div style="width: 100%; height: 10px; background-color: green;"></div>	3/4/2024, 5:45:07 PM	48 seconds	  	
> PLR 3f83247d-9e3b-4987-b69b-abf9d59c5302	-	✔ Succeeded	<div style="width: 100%; height: 10px; background-color: green;"></div>	3/4/2024, 5:43:13 PM	1 minute 14 seconds	  	
> PLR 01ab329a-94e8-4860-9cb9-94d471a598e1	 3  30  97  81	✔ Succeeded	<div style="width: 100%; height: 10px; background-color: green;"></div>	3/4/2024, 4:01:00 PM	5 minutes 28 seconds	  	

5 rows ▾ 1-3 of 3 < >

3. Click the expand row button besides PipelineRun name in the list to view the PipelineRun visualization. The pipeline run resource includes tasks to complete. When you hover the mouse pointer on a task card, you can view the steps to complete that particular task.

Pipeline Runs							Search
NAME	VULNERABILITIES	STATUS	TASK STATUS	STARTED	DURATION	ACTIONS	
> PLR 4ac96b4d-6cfa-4154-b2bc-27e27b4df977	-	✔ Succeeded	<div style="width: 100%; height: 10px; background-color: green;"></div>	3/4/2024, 5:45:07 PM	48 seconds	  	
> PLR 3f83247d-9e3b-4987-b69b-abf9d59c5302	-	✔ Succeeded	<div style="width: 100%; height: 10px; background-color: green;"></div>	3/4/2024, 5:43:13 PM	1 minute 14 seconds	  	
▾ PLR 01ab329a-94e8-4860-9cb9-94d471a598e1	 3  30  97  81	✔ Succeeded	<div style="width: 100%; height: 10px; background-color: green;"></div>	3/4/2024, 4:01:00 PM	5 minutes 28 seconds	  	



5 rows ▾ 1-3 of 3 < >

CHAPTER 7. MANAGING TEMPLATES

A template is a form composed of different UI fields that is defined in a YAML file. Templates include *actions*, which are steps that are executed in sequential order and can be executed conditionally.

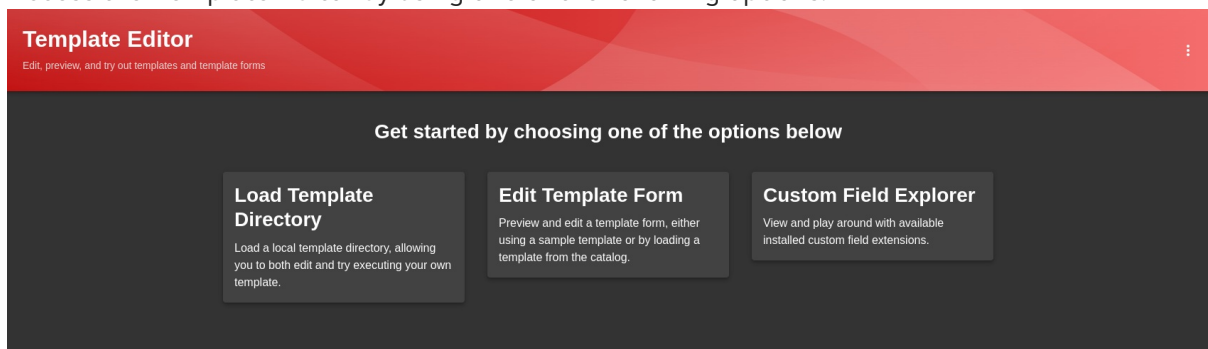
You can use templates to easily create Red Hat Developer Hub components, and then publish these components to different locations, such as the Red Hat Developer Hub software catalog, or repositories in GitHub or GitLab.

7.1. CREATING A TEMPLATE BY USING THE TEMPLATE EDITOR

You can create a template by using the Template Editor.

Procedure

1. Access the Template Editor by using one of the following options:



- Open the URL **`https://<rhdh_url>/create/edit`** for your Red Hat Developer Hub instance.
 - Click **Create...** in the navigation menu of the Red Hat Developer Hub console, then click the overflow menu button and select **Template editor**.
2. Click **Edit Template Form**
 3. Optional: Modify the YAML definition for the parameters of your template. For more information about these parameters, see [Section 7.2, “Creating a template as a YAML file”](#).
 4. In the **Name *** field, enter a unique name for your template.
 5. From the **Owner** drop-down menu, choose an owner for the template.
 6. Click **Next**.
 7. In the **Repository Location** view, enter the following information about the hosted repository that you want to publish the template to:
 - a. Select an available **Host** from the drop-down menu.



NOTE

Available hosts are defined in the YAML parameters by the **allowedHosts** field:

Example YAML

```
# ...
  ui:options:
    allowedHosts:
      - github.com
# ...
```

- b. In the **Owner *** field, enter an organization, user or project that the hosted repository belongs to.
 - c. In the **Repository *** field, enter the name of the hosted repository.
 - d. Click **Review**.
8. Review the information for accuracy, then click **Create**.

Verification

1. Click the **Catalog** tab in the navigation panel.
2. In the **Kind** drop-down menu, select **Template**.
3. Confirm that your template is shown in the list of existing templates.

7.2. CREATING A TEMPLATE AS A YAML FILE

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.

Template object example

```
apiVersion: scaffolder.backstage.io/v1beta3
kind: Template
metadata:
  name: template-name 1
  title: Example template 2
  description: An example template for v1beta3 scaffolder. 3
spec:
  owner: backstage/techdocs-core 4
  type: service 5
  parameters: 6
    - title: Fill in some steps
      required:
        - name
      properties:
        name:
```

```

    title: Name
    type: string
    description: Unique name of the component
  owner:
    title: Owner
    type: string
    description: Owner of the component
- title: Choose a location
  required:
  - repoUrl
  properties:
    repoUrl:
      title: Repository Location
      type: string
  steps: 7
  - id: fetch-base
    name: Fetch Base
    action: fetch:template
    # ...
  output: 8
  links:
  - title: Repository 9
    url: ${{ steps['publish'].output.remoteUrl }}
  - title: Open in catalog 10
    icon: catalog
    entityRef: ${{ steps['register'].output.entityRef }}
  # ...

```

- 1 Specify a name for the template.
- 2 Specify a title for the template. This is the title that is visible on the template tile in the **Create...** view.
- 3 Specify a description for the template. This is the description that is visible on the template tile in the **Create...** view.
- 4 Specify the ownership of the template. The **owner** field provides information about who is responsible for maintaining or overseeing the template within the system or organization. In the provided example, the **owner** field is set to **backstage/techdocs-core**. This means that this template belongs to the **techdocs-core** project in the **backstage** namespace.
- 5 Specify the component type. Any string value is accepted for this required field, but your organization should establish a proper taxonomy for these. Red Hat Developer Hub instances may read this field and behave differently depending on its value. For example, a **website** type component may present tooling in the Red Hat Developer Hub interface that is specific to just websites.

The following values are common for this field:

service

A backend service, typically exposing an API.

website

A website.

library

A software library, such as an npm module or a Java library.

- 6 Use the **parameters** section to specify parameters for user input that are shown in a form view when a user creates a component by using the template in the Red Hat Developer Hub console.
- 7 Use the **steps** section to specify steps that are executed in the backend. These steps must be defined by using a unique step ID, a name, and an action. You can view actions that are available on your Red Hat Developer Hub instance by visiting the URL **https://<rhdh_url>/create/actions**.
- 8 Use the **output** section to specify the structure of output data that is created when the template is used. The **output** section, particularly the **links** subsection, provides valuable references and URLs that users can utilize to access and interact with components that are created from the template.
- 9 Provides a reference or URL to the repository associated with the generated component.
- 10 Provides a reference or URL that allows users to open the generated component in a catalog or directory where various components are listed.

Additional resources

- [Backstage documentation - Writing Templates](#)
- [Backstage documentation - Builtin actions](#)
- [Backstage documentation - Writing Custom Actions](#)

7.3. IMPORTING AN EXISTING TEMPLATE TO RED HAT DEVELOPER HUB

You can add an existing template to your Red Hat Developer Hub instance by using the Catalog Processor.

Prerequisites

- You have created a directory or repository that contains at least one template YAML file.
- If you want to use a template that is stored in a repository such as GitHub or GitLab, you must configure a Red Hat Developer Hub integration for your provider.

Procedure

- In the **app-config.yaml** configuration file, modify the **catalog.rules** section to include a rule for templates, and configure the **catalog.locations** section to point to the template that you want to add, as shown in the following example:

```
# ...
catalog:
  rules:
    - allow: [Template] 1
  locations:
    - type: url 2
      target: https://<repository_url>/example-template.yaml 3
# ...
```

- 1 To allow new templates to be added to the catalog, you must add a **Template** rule.

- 2 If you are importing templates from a repository, such as GitHub or GitLab, use the **url** type.
- 3 Specify the URL for the template.

Verification

1. Click the **Catalog** tab in the navigation panel.
2. In the **Kind** drop-down menu, select **Template**.
3. Confirm that your template is shown in the list of existing templates.

Additional resources

- [Configuring a GitHub App in Developer Hub](#)
- [Enabling the GitLab OAuth authentication provider](#)