Red Hat Developer Hub 1.2

Administration guide for Red Hat Developer Hub

Last Updated: 2024-07-02

Red Hat Developer Hub 1.2 Administration guide for Red Hat Developer Hub

Legal Notice

Abstract

Red Hat Developer Hub is an enterprise-grade platform for building developer portals. As an administrative user, you can manage roles and permissions of other users and configure Developer Hub to meet the specific needs of your organization.

Table of Contents

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)

RHOCP is currently supported from version 4.12 to 4.15. See also the Red Hat Developer Hub Life Cycle .

Containers are available for the following CPU architectures:

- AMD64 and Intel 64 (x86_64)

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 **rhdh-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 OPENSHIFT CONTAINER PLATFORM

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

- The Helm chart
- The Red Hat Developer Hub Operator

2.1. DEPLOYING RED HAT DEVELOPER HUB ON OPENSHIFT 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**).

6. 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**

     a. 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**

     a. 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
```

7. Edit the other values if needed.

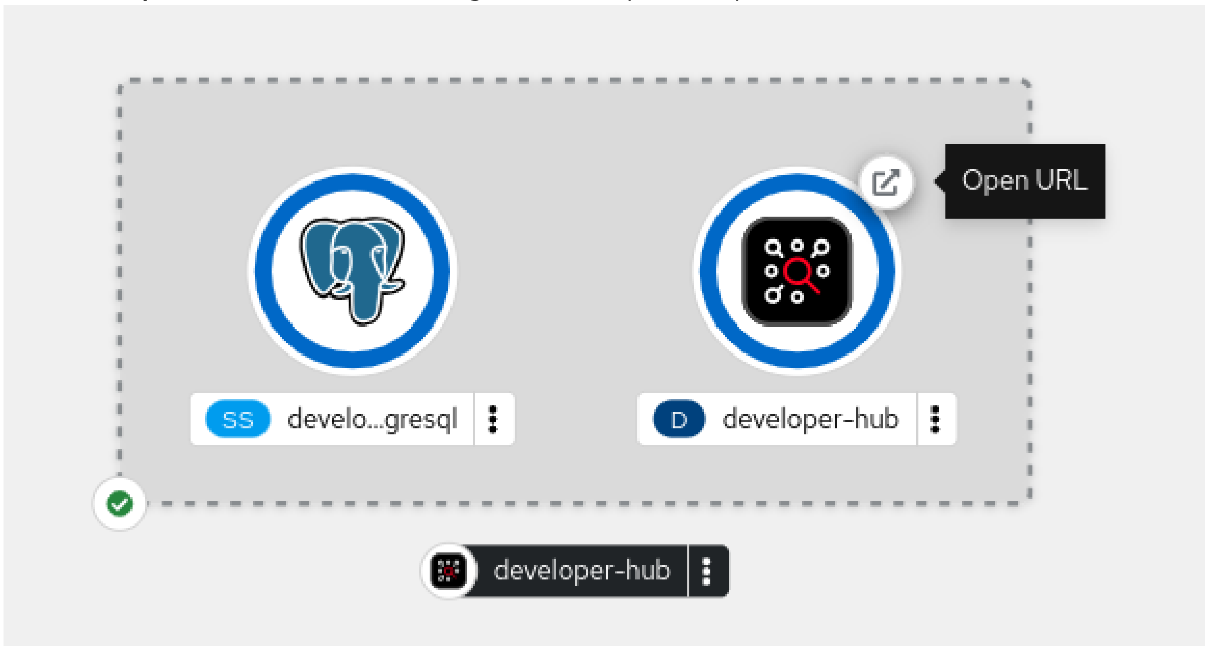
NOTE

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.

8. Click **Create** and wait for the database and Developer Hub to start.

9. 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 on Red Hat OpenShift Container Platform with Helm CLI

You can use the Helm CLI to install Red Hat Developer Hub on Red Hat OpenShift Container Platform.

Prerequisites

- You have installed the OpenShift CLI (**oc**) on your workstation.
- 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](https://docs.openshift.com/container-platform/4.15/authentication/using-rbac.html).
- 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](https://docs.openshift.com/container-platform/4.15/applications/projects/working-with-projects.html#odc-creating-projects-using-developer-perspective_projects).
- You have installed the Helm CLI.

Procedure

1. Create and activate the *<rhdh>* OpenShift Container Platform project:

```
NAMESPACE=<emphasis><rhdh></emphasis>
oc new-project ${NAMESPACE} || oc project ${NAMESPACE}
```

2. Install the Red Hat Developer Hub Helm chart:

```
helm upgrade redhat-developer-hub -i https://github.com/openshift-helm-
charts/charts/releases/download/redhat-redhat-developer-hub-1.2.1/redhat-developer-hub-
1.2.1.tgz
```

3. Configure your Developer Hub Helm chart instance with the Developer Hub database password and router base URL values from your OpenShift Container Platform cluster:

```
PASSWORD=$(oc get secret redhat-developer-hub-postgresql -o jsonpath="
{.data.password}" | base64 -d)
CLUSTER_ROUTER_BASE=$(oc get route console -n openshift-console -
o=jsonpath='{.spec.host}' | sed 's/^[^.]*\.//')
helm upgrade redhat-developer-hub -i "https://github.com/openshift-helm-
charts/charts/releases/download/redhat-redhat-developer-hub-1.2.1/redhat-developer-hub-
1.2.1.tgz" \
  --set global.clusterRouterBase="$CLUSTER_ROUTER_BASE" \
  --set global.postgresql.auth.password="$PASSWORD"
```

4. Display the running Developer Hub instance URL:

```
echo "https://redhat-developer-hub-$NAMESPACE.$CLUSTER_ROUTER_BASE"
```

Verification

- Open the running Developer Hub instance URL in your browser to use Developer Hub.

2.1.2. Adding a custom application configuration file to OpenShift Container Platform using the Helm chart

You can use the Red Hat Developer Hub Helm chart to add a custom application configuration file to your OpenShift Container Platform instance.

Prerequisites

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

Procedure

1. From the OpenShift Container Platform web console, select the **ConfigMaps** tab.
2. Click **Create ConfigMap**.
3. From **Create ConfigMap** page, select the **YAML view** option in **Configure via** and make changes to the file, if needed.
4. Click **Create**.
5. Go to the **Helm** tab to see the list of Helm releases.
6. Click the overflow menu on the Helm release that you want to use and select **Upgrade**.
7. Use either the **Form view** or **YAML view** to edit the Helm configuration.
   - Using **Form view**
     a. Expand **Root Schema → Backstage chart schema → Backstage parameters → Extra app configuration files to inline into command arguments**.
     b. Click the **Add Extra app configuration files to inline into command arguments** link.
     c. Enter the value in the following fields:
        - **configMapRef**: **app-config-rhdh**

- filename: **app-config-rhdh.yaml**

d. Click **Upgrade**.

- Using **YAML view**

a. Set the value of the **upstream.backstage.extraAppConfig.configMapRef** and **upstream.backstage.extraAppConfig.filename** parameters as follows:

```
# ... other Red Hat Developer Hub Helm Chart configurations
upstream:
  backstage:
    extraAppConfig:
      - configMapRef: app-config-rhdh
        filename: app-config-rhdh.yaml
# ... other Red Hat Developer Hub Helm Chart configurations
```

b. Click **Upgrade**.

2.1.3. 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 OpenShift CLI (**oc**) 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 OpenShift CLI (**oc**), 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 OPENSHIFT 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

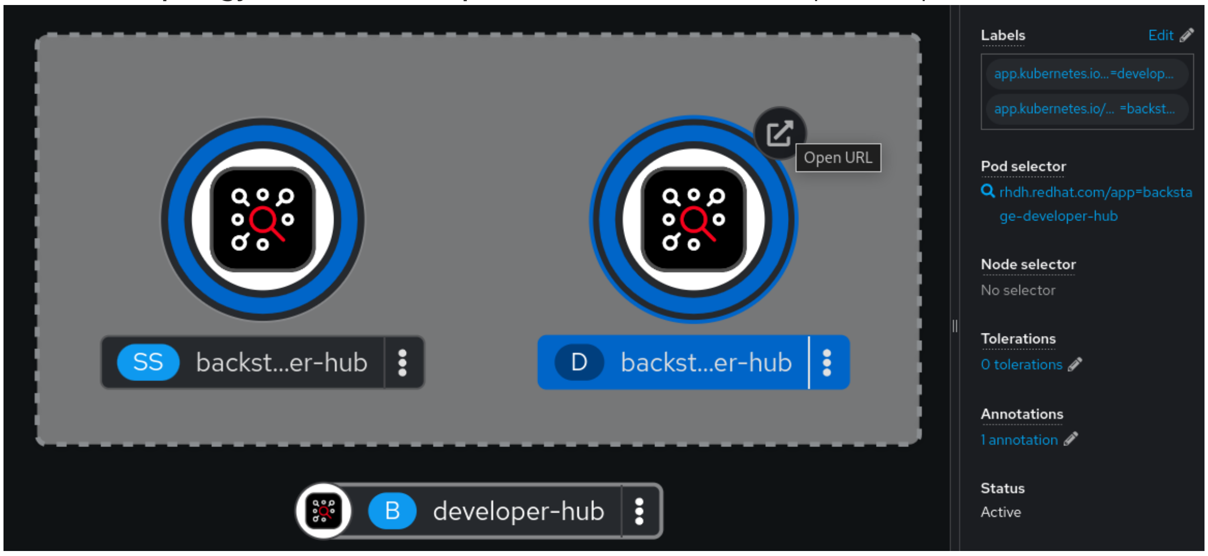
For more information about creating a project in OpenShift Container Platform, see Creating a project by using the web console in the Red Hat OpenShift Container Platform documentation.

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.



Labels Edit
app.kubernetes.io...=develop...
app.kubernetes.io/... =backst...
Pod selector
rhdh.redhat.com/app=backstage-developer-hub
Node selector
No selector
Tolerations
0 tolerations
Annotations
1 annotation
Status
Active

Additional resources

- OpenShift Container Platform - Building applications overview

2.2.1. Adding a custom application configuration file to OpenShift Container Platform using the Operator

A custom application configuration file is a **ConfigMap** object that you can use to change the configuration of your Red Hat Developer Hub instance. If you are deploying your Developer Hub instance on Red Hat OpenShift Container Platform, you can use the Red Hat Developer Hub Operator to add a custom application configuration file to your OpenShift Container Platform instance by creating the **ConfigMap** object and referencing it in the Developer Hub custom resource (CR).

The custom application configuration file contains a sensitive environment variable, named **BACKEND_SECRET**. This variable contains a mandatory backend authentication key that Developer

Hub uses to reference an environment variable defined in an OpenShift Container Platform secret. You must create a secret, named 'secrets-rhdh', and reference it in the Developer Hub CR.

NOTE

You are responsible for protecting your Red Hat Developer Hub installation from external and unauthorized access. Manage the backend authentication 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.
- Your administrator has installed the Red Hat Developer Hub Operator in OpenShift Container Platform. For more information, see Installing the Red Hat Developer Hub Operator .
- You have created the Red Hat Developer Hub CR in OpenShift Container Platform.

Procedure

1. From the **Developer** perspective in the OpenShift Container Platform web console, select the **Topology** view, and click the **Open URL** icon on the Developer Hub pod to identify your Developer Hub external URL: *<RHDH_URL>*.
2. From the **Developer** perspective in the OpenShift Container Platform web console, select the **ConfigMaps** view.
3. Click **Create ConfigMap**.
4. Select the **YAML view** option in **Configure via** and use the following example as a base template to create a **ConfigMap** object, 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: <RHDH_URL> 1
    backend:
      auth:
        keys:
          - secret: "${BACKEND_SECRET}" 2
      baseUrl: <RHDH_URL> 3
      cors:
        origin: <RHDH_URL> 4
```

1 Set the external URL of your Red Hat Developer Hub instance.

2 Use an environment variable exposing an OpenShift Container Platform secret to define the mandatory Developer Hub backend authentication key.

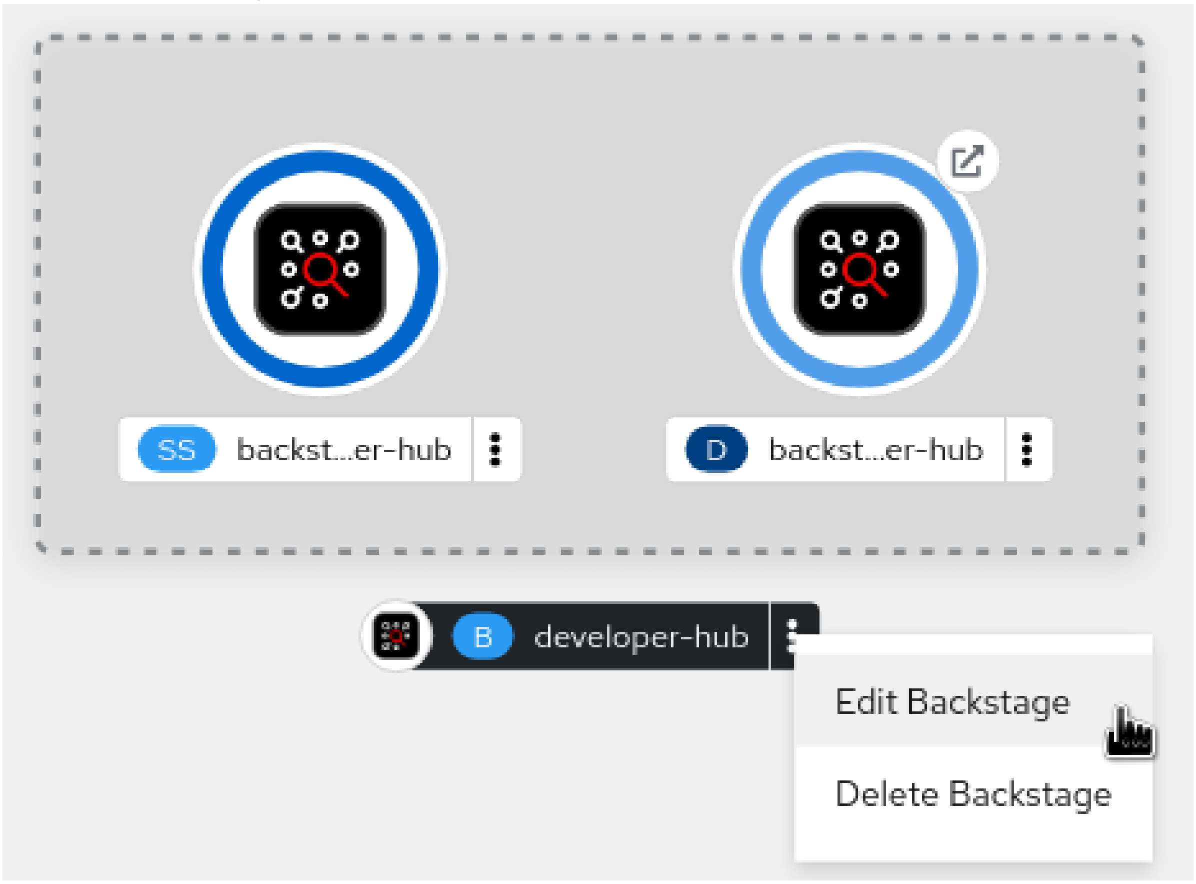
3 Set the external URL of your Red Hat Developer Hub instance.

4 Set the external URL of your Red Hat Developer Hub instance.

5. Click **Create**.
6. Select the **Secrets** view.
7. Click **Create Key/value Secret**
8. Create a secret named **secrets-rhdh**.
9. 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")'
```

10. Click **Create**.
11. Select the **Topology** view.
12. 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.



13. In the CR, enter the name of the custom application configuration config map as the value for the **spec.application.appConfig.configMaps** field, and enter the name of your secret as the value for the **spec.application.extraEnvs.secrets** field. For example:

```
apiVersion: v1
kind: ConfigMap
```

```
metadata:
  name: 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
```

14. Click **Save**.

15. Navigate back to the **Topology** view and wait for the Red Hat Developer Hub pod to start.

16. Click the **Open URL** icon to use the Red Hat Developer Hub platform with the 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.

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.

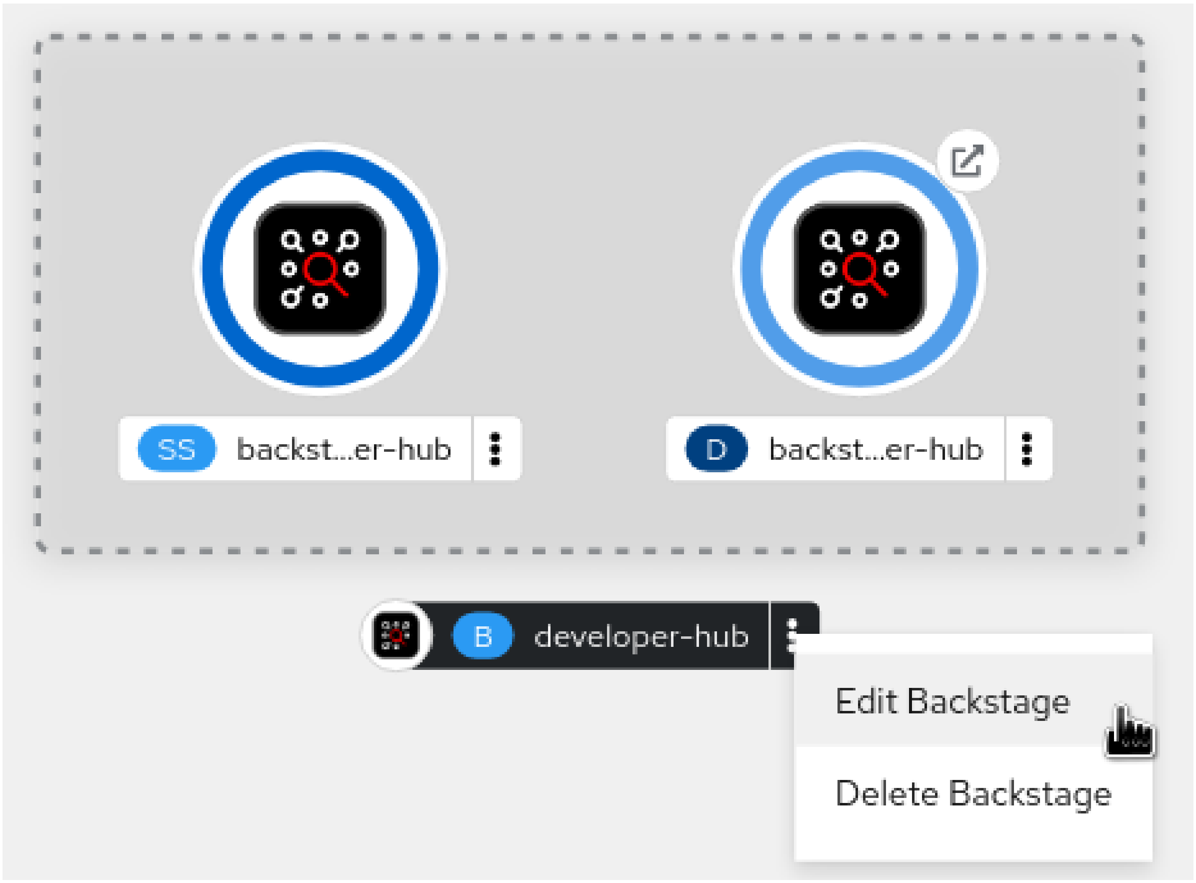
Example ConfigMap object using the GitHub dynamic plugin

```
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** view.

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 OpenShift CLI (**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. CONFIGURING EXTERNAL POSTGRESQL DATABASES

As an administrator, you can configure and use external PostgreSQL databases in Red Hat Developer Hub. You can use a PostgreSQL certificate file to configure an external PostgreSQL instance using the Operator or Helm Chart.

NOTE

Developer Hub supports only configuring external PostgreSQL databases. You can perform maintenance activities, such as backing up your data or configuring high availability (HA) for the external PostgreSQL databases.

Also, configuring an external PostgreSQL instance by using the Red Hat Developer Hub Operator or Helm Chart is not intended for production use.

3.1. CONFIGURING AN EXTERNAL POSTGRESQL INSTANCE USING THE OPERATOR

You can configure an external PostgreSQL instance using the Red Hat Developer Hub Operator. By default, the Operator creates and manages a local instance of PostgreSQL in the same namespace where you have deployed the RHDH instance. However, you can change this default setting to configure an external PostgreSQL database server, for example, Amazon Web Services (AWS) Relational Database Service (RDS) or Azure database.

Prerequisites

- You are using a supported version of PostgreSQL. For more information, see the Product life cycle page.
- You have the following details:
    - **db-host**: Denotes your PostgreSQL instance Domain Name System (DNS) or IP address
    - **db-port**: Denotes your PostgreSQL instance port number, such as **5432**
    - **username**: Denotes the user name to connect to your PostgreSQL instance
    - **password**: Denotes the password to connect to your PostgreSQL instance
- You have installed the Red Hat Developer Hub Operator.
- Optional: You have a CA certificate, Transport Layer Security (TLS) private key, and TLS certificate so that you can secure your database connection by using the TLS protocol. For more information, refer to your PostgreSQL vendor documentation.

NOTE

By default, Developer Hub uses a database for each plugin and automatically creates it if none is found. You might need the **Create Database** privilege in addition to **PSQL Database** privileges for configuring an external PostgreSQL instance.

Procedure

1. Optional: Create a certificate secret to configure your PostgreSQL instance with a TLS connection:

```
cat <<EOF | oc -n <your-namespace> create -f -
apiVersion: v1
kind: Secret
metadata:
 name: <crt-secret> 1
type: Opaque
stringData:
 postgres-ca.pem: |-
  -----BEGIN CERTIFICATE-----
  <ca-certificate-key> 2
 postgres-key.key: |-
  -----BEGIN CERTIFICATE-----
  <tls-private-key> 3
 postgres-crt.pem: |-
  -----BEGIN CERTIFICATE-----
  <tls-certificate-key> 4
  # ...
EOF
```

1 Provide the name of the certificate secret.

2 Provide the CA certificate key.

3 Optional: Provide the TLS private key.

4 Optional: Provide the TLS certificate key.

2. Create a credential secret to connect with the PostgreSQL instance:

```
cat <<EOF | oc -n <your-namespace> create -f -
apiVersion: v1
kind: Secret
metadata:
 name: <cred-secret> 1
type: Opaque
stringData: 2
 POSTGRES_PASSWORD: <password>
 POSTGRES_PORT: "<db-port>"
 POSTGRES_USER: <username>
 POSTGRES_HOST: <db-host>
 PGSSLMODE: <ssl-mode> # for TLS connection 3
 NODE_EXTRA_CA_CERTS: <abs-path-to-pem-file> # for TLS connection, e.g. /opt/app-
root/src/postgres-crt.pem 4
EOF
```

1 Provide the name of the credential secret.

2 Provide credential data to connect with your PostgreSQL instance.

3 Optional: Provide the value based on the required Secure Sockets Layer (SSL) mode.

4 Optional: Provide the value only if you need a TLS connection for your PostgreSQL instance.

3. Create a **Backstage** custom resource (CR):

```
cat <<EOF | oc -n <your-namespace> create -f -
apiVersion: rhdh.redhat.com/v1alpha1
kind: Backstage
metadata:
  name: <backstage-instance-name>
spec:
  database:
    enableLocalDb: false 1
  application:
    extraFiles:
      mountPath: <path> # e g /opt/app-root/src
      secrets:
        - name: <crt-secret> 2
          key: postgres-crt.pem, postgres-ca.pem, postgres-key.key # key name as in <crt-
secret> Secret
    extraEnvs:
      secrets:
        - name: <cred-secret> 3
        # ...
```

1 Set the value of the **enableLocalDb** parameter to **false** to disable creating local PostgreSQL instances.

2 Provide the name of the certificate secret if you have configured a TLS connection.

3 Provide the name of the credential secret that you created.

NOTE

The environment variables listed in the **Backstage** CR work with the Operator default configuration. If you have changed the Operator default configuration, you must reconfigure the **Backstage** CR accordingly.

4. Apply the **Backstage** CR to the namespace where you have deployed the RHDH instance.

3.2. CONFIGURING AN EXTERNAL POSTGRESQL INSTANCE USING THE HELM CHART

You can configure an external PostgreSQL instance by using the Helm Chart. By default, the Helm Chart creates and manages a local instance of PostgreSQL in the same namespace where you have deployed the RHDH instance. However, you can change this default setting to configure an external PostgreSQL database server, for example, Amazon Web Services (AWS) Relational Database Service (RDS) or Azure database.

Prerequisites

- You are using a supported version of PostgreSQL. For more information, see the Product life cycle page.
- You have the following details:
  - **db-host**: Denotes your PostgreSQL instance Domain Name System (DNS) or IP address
  - **db-port**: Denotes your PostgreSQL instance port number, such as **5432**
  - **username**: Denotes the user name to connect to your PostgreSQL instance
  - **password**: Denotes the password to connect to your PostgreSQL instance
- You have installed the RHDH application by using the Helm Chart.
- Optional: You have a CA certificate, Transport Layer Security (TLS) private key, and TLS certificate so that you can secure your database connection by using the TLS protocol. For more information, refer to your PostgreSQL vendor documentation.

NOTE

By default, Developer Hub uses a database for each plugin and automatically creates it if none is found. You might need the **Create Database** privilege in addition to **PSQL Database** privileges for configuring an external PostgreSQL instance.

Procedure

1. Optional: Create a certificate secret to configure your PostgreSQL instance with a TLS connection:

```
cat <<EOF | oc -n <your-namespace> create -f -
apiVersion: v1
kind: Secret
metadata:
 name: <crt-secret> 1
type: Opaque
stringData:
 postgres-ca.pem: |-
  -----BEGIN CERTIFICATE-----
  <ca-certificate-key> 2
 postgres-key.key: |-
  -----BEGIN CERTIFICATE-----
  <tls-private-key> 3
 postgres-crt.pem: |-
  -----BEGIN CERTIFICATE-----
  <tls-certificate-key> 4
  # ...
EOF
```

1 Provide the name of the certificate secret.

2 Provide the CA certificate key.

3 Optional: Provide the TLS private key.

4 Optional: Provide the TLS certificate key.

2. Create a credential secret to connect with the PostgreSQL instance:

```
cat <<EOF | oc -n <your-namespace> create -f -
apiVersion: v1
kind: Secret
metadata:
 name: <cred-secret> 1
type: Opaque
stringData: 2
 POSTGRES_PASSWORD: <password>
 POSTGRES_PORT: "<db-port>"
 POSTGRES_USER: <username>
 POSTGRES_HOST: <db-host>
 PGSSLMODE: <ssl-mode> # for TLS connection 3
 NODE_EXTRA_CA_CERTS: <abs-path-to-pem-file> # for TLS connection, e.g. /opt/app-
root/src/postgres-crt.pem 4
EOF
```

1 Provide the name of the credential secret.

2 Provide credential data to connect with your PostgreSQL instance.

3 Optional: Provide the value based on the required Secure Sockets Layer (SSL) mode.

4 Optional: Provide the value only if you need a TLS connection for your PostgreSQL instance.

3. Configure your PostgreSQL instance in the Helm configuration file named **values.yaml**:

```
# ...
upstream:
  postgresql:
    enabled: false  # disable PostgreSQL instance creation 1
    auth:
      existingSecret: <cred-secret> # inject credentials secret to Backstage 2
  backstage:
    appConfig:
      backend:
        database:
          connection:  # configure Backstage DB connection parameters
            host: ${POSTGRES_HOST}
            port: ${POSTGRES_PORT}
            user: ${POSTGRES_USER}
            password: ${POSTGRES_PASSWORD}
            ssl:
              rejectUnauthorized: true,
              ca:
                $file: /opt/app-root/src/postgres-ca.pem
              key:
                $file: /opt/app-root/src/postgres-key.key
              cert:
                $file: /opt/app-root/src/postgres-crt.pem
  extraEnvVarsSecrets:
    - <cred-secret> # inject credentials secret to Backstage 3
```

```
extraEnvVars:
  - name: BACKEND_SECRET
    valueFrom:
      secretKeyRef:
        key: backend-secret
        name: '{{ include "janus-idp.backend-secret-name" $ }}'
extraVolumeMounts:
  - mountPath: /opt/app-root/src/dynamic-plugins-root
    name: dynamic-plugins-root
  - mountPath: /opt/app-root/src/postgres-crt.pem
    name: postgres-crt # inject TLS certificate to Backstage cont. (4)
    subPath: postgres-crt.pem
  - mountPath: /opt/app-root/src/postgres-ca.pem
    name: postgres-ca # inject CA certificate to Backstage cont. (5)
    subPath: postgres-ca.pem
  - mountPath: /opt/app-root/src/postgres-key.key
    name: postgres-key # inject TLS private key to Backstage cont. (6)
    subPath: postgres-key.key
extraVolumes:
  - ephemeral:
      volumeClaimTemplate:
        spec:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 1Gi
    name: dynamic-plugins-root
  - configMap:
      defaultMode: 420
      name: dynamic-plugins
      optional: true
    name: dynamic-plugins
  - name: dynamic-plugins-npmrc
    secret:
      defaultMode: 420
      optional: true
      secretName: dynamic-plugins-npmrc
  - name: postgres-crt
    secret:
      secretName: <crt-secret> (7)
      # ...
```

1. Set the value of the **upstream.postgresql.enabled** parameter to **false** to disable creating local PostgreSQL instances.
2. Provide the name of the credential secret.
3. Provide the name of the credential secret.
4. Optional: Provide the name of the TLS certificate only for a TLS connection.
5. Optional: Provide the name of the CA certificate only for a TLS connection.
6. Optional: Provide the name of the TLS private key only if your TLS connection requires a private key.

7 Provide the name of the certificate secret if you have configured a TLS connection.

4. Apply the configuration changes in your Helm configuration file named **values.yaml**:

```
helm upgrade -n <your-namespace> <your-deploy-name> openshift-helm-charts/redhat-developer-hub -f values.yaml --version 1.2.1
```

3.3. MIGRATING LOCAL DATABASES TO AN EXTERNAL DATABASE SERVER USING THE OPERATOR

By default, Red Hat Developer Hub hosts the data for each plugin in a PostgreSQL database. When you fetch the list of databases, you might see multiple databases based on the number of plugins configured in Developer Hub. You can migrate the data from an RHDH instance hosted on a local PostgreSQL server to an external PostgreSQL service, such as AWS RDS, Azure database, or Crunchy database. To migrate the data from each RHDH instance, you can use PostgreSQL utilities, such as **pg_dump** with **psql** or **pgAdmin**.

NOTE

The following procedure uses a database copy script to do a quick migration.

Prerequisites

- You have installed the **pg_dump** and **psql** utilities on your local machine.
- For data export, you have the PGSQL user privileges to make a full dump of local databases.
- For data import, you have the PGSQL admin privileges to create an external database and populate it with database dumps.

Procedure

1. Configure port forwarding for the local PostgreSQL database pod by running the following command on a terminal:

```
oc port-forward -n <your-namespace> <pgsql-pod-name> <forward-to-port>:<forward-from-port>
```

Where:

- The **<pgsql-pod-name>** variable denotes the name of a PostgreSQL pod with the format **backstage-psql-<deployment-name>-<_index>**.
- The **<forward-to-port>** variable denotes the port of your choice to forward PostgreSQL data to.
- The **<forward-from-port>** variable denotes the local PostgreSQL instance port, such as **5432**.

Example: Configuring port forwarding

```
oc port-forward -n developer-hub backstage-psql-developer-hub-0 15432:5432
```

2. Make a copy of the following **db_copy.sh** script and edit the details based on your configuration:

```
#!/bin/bash

to_host=<db-service-host> (1)
to_port=5432 (2)
to_user=postgres (3)

from_host=127.0.0.1 (4)
from_port=15432 (5)
from_user=postgres (6)

allDB=("backstage_plugin_app" "backstage_plugin_auth" "backstage_plugin_catalog"
"backstage_plugin_permission" "backstage_plugin_scaffolder" "backstage_plugin_search")
(7)

for db in ${!allDB[@]};
do
  db=${allDB[$db]}
  echo Copying database: $db
  PGPASSWORD=$TO_PSW psql -h $to_host -p $to_port -U $to_user -c "create database
$db;"
  pg_dump -h $from_host -p $from_port -U $from_user -d $db | PGPASSWORD=$TO_PSW
psql -h $to_host -p $to_port -U $to_user -d $db
done
```

1. The destination host name, for example, **<db-instance-name>.rds.amazonaws.com**.
2. The destination port, such as **5432**.
3. The destination server username, for example, **postgres**.
4. The source host name, such as **127.0.0.1**.
5. The source port number, such as the **<forward-to-port>** variable.
6. The source server username, for example, **postgres**.
7. The name of databases to import in double quotes separated by spaces, for example, **("backstage_plugin_app" "backstage_plugin_auth" "backstage_plugin_catalog" "backstage_plugin_permission" "backstage_plugin_scaffolder" "backstage_plugin_search")**.

3. Create a destination database for copying the data:

```
/bin/bash TO_PSW=<destination-db-password> /path/to/db_copy.sh (1)
```

1. The **<destination-db-password>** variable denotes the password to connect to the destination database.

> **NOTE**
>
> You can stop port forwarding when the copying of the data is complete. For more information about handling large databases and using the compression tools, see the Handling Large Databases section on the PostgreSQL website.

4. Reconfigure your **Backstage** custom resource (CR). For more information, see Configuring an external PostgreSQL instance using the Operator.

5. Check that the following code is present at the end of your **Backstage** CR after reconfiguration:

```
# ...
spec:
  database:
    enableLocalDb: false
  application:
  # ...
    extraFiles:
      secrets:
        - name: <crt-secret>
          key: postgres-crt.pem # key name as in <crt-secret> Secret
    extraEnvs:
      secrets:
        - name: <cred-secret>
# ...
```

> **NOTE**
>
> Reconfiguring the **Backstage** CR deletes the corresponding **StatefulSet** and **Pod** objects, but does not delete the **PersistenceVolumeClaim** object. Use the following command to delete the local **PersistenceVolumeClaim** object:
>
> ```
> oc -n developer-hub delete pvc <local-psql-pvc-name>
> ```
>
> where, the **<local-psql-pvc-name>** variable is in the **data-<psql-pod-name>** format.

6. Apply the configuration changes.

Verification

1. Verify that your RHDH instance is running with the migrated data and does not contain the local PostgreSQL database by running the following command:

```
oc get pods -n <your-namespace>
```

2. Check the output for the following details:

   - The **backstage-developer-hub-xxx** pod is in running state.
   - The **backstage-psql-developer-hub-0** pod is not available.
     You can also verify these details using the **Topology** view in the OpenShift Container Platform web console.

CHAPTER 4. ENABLING AUTHENTICATION IN RED HAT DEVELOPER HUB

Authentication within Red Hat Developer Hub facilitates user sign-in, identification, and access to external resources. It supports multiple authentication providers.

Authentication providers are typically used in the following ways:

- One provider for sign-in and identification.
- Additional providers for accessing external resources.

The Red Hat Developer Hub supports the following authentication providers:

Microsoft Azure	**microsoft**
GitHub	**github**
Keycloak	**oidc**

For each provider that you want to use, follow the dedicated procedure to complete the following tasks:

1. Set up the shared secret that the authentication provider and Red Hat Developer Hub require to communicate.
2. Configure Red Hat Developer Hub to use the authentication provider.

4.1. ENABLING THE MICROSOFT AZURE AUTHENTICATION PROVIDER

Red Hat Developer Hub includes a Microsoft Azure authentication provider that can authenticate users by using OAuth.

Procedure

1. To allow Developer Hub to authenticate with Microsoft Azure, create an OAuth Application in Microsoft Azure.
   a. Go to Azure Portal > App registrations, and create an **App Registration** for Developer Hub.
   b. On your **App registration** overview page, add a new **Web platform configuration**, with the configuration:

      Redirect URI
         Enter the backend authentication URI set in Developer Hub: **https://*<APP_FQDN>*/api/auth/microsoft/handler/frame**

      Front-channel logout URL
         Leave blank.

      Implicit grant and hybrid flows
         Leave all checkboxes cleared.

c. On the **API permissions** tab, click **Add Permission**, then add the following **Delegated permission** for the **Microsoft Graph API**:

email, offline_access, openid, profile, User.Read, (Optional)
: Optional custom scopes of the Microsoft Graph API that you define both here and in the Developer Hub configuration (**app-config-rhdh.yaml**).

> **NOTE**
>
> Your company might require you to grant admin consent for these permissions. Even if your company does not require admin consent, you might do so as it means users do not need to individually consent the first time they access backstage. To grant admin consent, a directory admin must go to the admin consent page and click **Grant admin consent for COMPANY NAME**.

a. Go to the **Certificates & Secrets** page, then the **Client secrets** tab, and create a new client secret. Save the **Client secret** for the next step.

1. Add your Microsoft Azure credentials in your Developer Hub secrets.

b. Edit your Developer Hub secrets, such as **secrets-rhdh**.

c. Add the following key/value pairs:

- **AUTH_AZURE_CLIENT_ID**: Enter the **Application ID** that you generated on Microsoft Azure.
- **AUTH_AZURE_CLIENT_SECRET**: Enter the **Client secret** that you generated on Microsoft Azure.
- **AUTH_AZURE_TENANT_ID**: Enter your **Tenant ID** on Microsoft Azure.

1. Set up the Microsoft Azure authentication provider in your Developer Hub custom configuration.
   Edit your custom Developer Hub config map, such as **app-config-rhdh**.

   In the **app-config-rhdh.yaml** content, add the **microsoft** provider configuration under the root **auth** configuration, and enable the **microsoft** provider for sign-in:

   app-config-rhdh.yaml fragment

```
auth:
  environment: production
  providers:
    microsoft:
      production:
        clientId: ${AUTH_AZURE_CLIENT_ID}
        clientSecret: ${AUTH_AZURE_CLIENT_SECRET}
        tenantId: ${AUTH_AZURE_TENANT_ID}
        # domainHint: ${AUTH_AZURE_TENANT_ID} (1)
        # additionalScopes: (2)
         # - Mail.Send
signInPage: microsoft (3)
```

1. Optional for single-tenant applications. You can reduce login friction for users with accounts in multiple tenants by automatically filtering out accounts from other tenants. If you want to use this parameter for a single-tenant application, uncomment and enter the tenant ID. If your application registration is multi-tenant, leave this parameter blank. For more information, see Home Realm Discovery.

2. Optional for additional scopes. To add scopes for the application registration, uncomment and enter the list of scopes that you want to add. The default and mandatory value is **['user.read']**.

3. To enable the Microsoft Azure provider as default sign-in provider.

NOTE

Optional for environments with restrictions on outgoing access, such as firewall rules. If your environment has outgoing access restrictions make sure your Backstage backend has access to the following hosts:

- **login.microsoftonline.com**: To get and exchange authorization codes and access tokens.
- **graph.microsoft.com**: To fetch user profile information (as seen in this source code). If this host is unreachable, users might see an *Authentication failed, failed to fetch user profile* error when they attempt to log in.

4.2. ENABLING THE GITLAB OAUTH AUTHENTICATION PROVIDER

Red Hat Developer Hub includes a GitLab authentication provider that can authenticate users by using GitLab OAuth.

Prerequistes

- You configured Developer Hub with a custom config map and secret.

Procedure

1. To allow Developer Hub to authenticate with Gitlab, create an OAuth Application in Gitlab. Go to GitLab User settings > Applications, and click the **Add new application** button.

   Name

   Enter your application name, such as **Developer Hub**.

   Redirect URI

   Enter the backend authentication URI set in Developer Hub, such as **http://*<APP_FQDN>*/api/auth/gitlab/handler/frame**. Due to a peculiarity with GitLab OAuth, ensure the URL has no trailing / after 'frame'.

   Scopes

   Select the following scopes from the list and click **Save application**:

   read_user

   Grants read-only access to the authenticated user’s profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.

read_repository
: Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.

write_repository
: Grants read/write access to repositories on private projects using Git-over-HTTP (not using the API).

openid
: Grants permission to authenticate with GitLab using OpenID Connect. Also gives read-only access to the user's profile and group memberships.

profile
: Grants read-only access to the user's profile data using OpenID Connect.

email
: Grants read-only access to the user's primary email address using OpenID Connect.

Save the **Application ID** and **Secret** for the next step.

2. Add your Gitlab credentials in your Developer Hub secrets.
   a. Edit your Developer Hub secrets, such as **secrets-rhdh**.
   b. Add the following key/value pairs:

      AUTH_GITLAB_CLIENT_ID
      : Enter the **Application ID** that you generated on GitLab, such as **4928c033ab3d592845c044a653bc20583baf84f2e67b954c6fdb32a532ab76c9**.

      AUTH_GITLAB_CLIENT_SECRET
      : Enter the **Secret** that you generated on Gitlab, such as **gloas-f2c9c350759cc08346fbf94a476ae83c579c76dd629fc5eeef9dc21eedfe0475**.

3. Set up the Gitlab authentication provider in your Developer Hub custom configuration.
   a. Edit your custom Developer Hub config map, such as **app-config-rhdh**.
   b. In the **app-config-rhdh.yaml** content, add the **gitlab** provider configuration under the root **auth** configuration, and enable the **gitlab** provider for sign-in:

      app-config-rhdh.yaml fragment

```
auth:
  environment: production
  providers:
    gitlab:
      production:
        clientId: ${AUTH_GITLAB_CLIENT_ID}
        clientSecret: ${AUTH_GITLAB_CLIENT_SECRET}
        # audience: https://gitlab.company.com (1)
        # callbackUrl: https://<APP_FQDN>/api/auth/gitlab/handler/frame (2)
signInPage: gitlab (3)
```

1. Optionally, when using a self-hosted Gitlab: uncomment, and enter your GitLab instance base URL, such as **https://gitlab.company.com**.

2. Optionally, when using a custom redirect URI: uncomment, and enter the URL matching the **Redirect URI** registered when creating your GitLab OAuth App, such as **http://*<APP_FQDN>*/api/auth/gitlab/handler/frame**. Due to a peculiarity with GitLab OAuth, ensure the URL has no trailing / after 'frame'.

3. To enable the Gitlab provider as default sign-in provider.

Verification

1. The **backstage-developer-hub** deployment starts a pod with the updated configuration.

2. Your Developer Hub sign-in page displays **Sign in using GitLab**.

CHAPTER 5. TELEMETRY DATA COLLECTION

The telemetry data collection feature is enabled by default. Red Hat Developer Hub sends telemetry data to Red Hat by using the **backstage-plugin-analytics-provider-segment** plugin.

IMPORTANT

You can disable the telemetry data collection feature based on your needs. For example, in an air-gapped environment, you can disable this feature to avoid needless outbound requests affecting the responsiveness of the RHDH application. For more details, see the Disabling telemetry data collection in RHDH section.

Red Hat collects and analyzes the following data to improve your experience with Red Hat Developer Hub:

- Events of page visits and clicks on links or buttons.
- System-related information, for example, locale, timezone, user agent including browser and OS details.
- Page-related information, for example, title, category, extension name, URL, path, referrer, and search parameters.
- Anonymized IP addresses, recorded as **0.0.0.0**.
- Anonymized username hashes, which are unique identifiers used solely to identify the number of unique users of the RHDH application.

With RHDH, you can customize the telemetry data collection feature and the telemetry Segment source configuration based on your needs.

5.1. DISABLING TELEMETRY DATA COLLECTION IN RHDH

To disable telemetry data collection, you must disable the **analytics-provider-segment** plugin either using the Helm Chart or the Red Hat Developer Hub Operator configuration.

NOTE

If the **analytics-provider-segment** plugin is already present in your dynamic plugins configuration, set the value of the **plugins.disabled** parameter to **true** to disable telemetry data collection.

Procedure

1. Disable the **analytics-provider-segment** plugin by using one of the following options:

   Using Helm Chart

   - Add the following YAML code in your Helm configuration file:

```
# ...
global:
  dynamic:
    plugins:
      - package: './dynamic-plugins/dist/janus-idp-backstage-plugin-analytics-provider-segment'
        disabled: true
# ...
```

Using the Operator

a. Perform one of the following steps:

- If you have created the **dynamic-plugins-rhdh** ConfigMap file, add the **analytics-provider-segment** plugin to the list of plugins and set its **plugins.disabled** parameter to **true**.
- If you have not created the ConfigMap file, create it with the following YAML code:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: dynamic-plugins-rhdh
data:
  dynamic-plugins.yaml: |
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: './dynamic-plugins/dist/janus-idp-backstage-plugin-analytics-provider-segment'
        disabled: true
```

b. Set the value of the **dynamicPluginsConfigMapName** parameter to the name of the ConfigMap file in your **Backstage** custom resource:

```
# ...
spec:
  application:
    dynamicPluginsConfigMapName: dynamic-plugins-rhdh
# ...
```

2. Save the configuration changes.

5.2. ENABLING TELEMETRY DATA COLLECTION IN RHDH

The telemetry data collection feature is enabled by default. However, if you have disabled the feature and want to re-enable it, you must enable the **analytics-provider-segment** plugin either by using the Helm Chart or the Red Hat Developer Hub Operator configuration.

NOTE

If the **analytics-provider-segment** plugin is already present in your dynamic plugins configuration, set the value of the **plugins.disabled** parameter to **false** to enable telemetry data collection.

Procedure

1. Configure the **analytics-provider-segment** plugin by using one of the following options:

Using Helm Chart

- Add the following YAML code in your Helm configuration file:

```
# ...
global:
  dynamic:
    plugins:
      - package: './dynamic-plugins/dist/janus-idp-backstage-plugin-analytics-provider-segment'
        disabled: false
# ...
```

Using the Operator

a. Perform one of the following steps:

- If you have created the **dynamic-plugins-rhdh** ConfigMap file, add the **analytics-provider-segment** plugin to the list of plugins and set its **plugins.disabled** parameter to **false**.

- If you have not created the ConfigMap file, create it with the following YAML code:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: dynamic-plugins-rhdh
data:
  dynamic-plugins.yaml: |
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: './dynamic-plugins/dist/janus-idp-backstage-plugin-analytics-provider-segment'
        disabled: false
```

b. Set the value of the **dynamicPluginsConfigMapName** parameter to the name of the ConfigMap file in your **Backstage** custom resource:

```
# ...
spec:
  application:
    dynamicPluginsConfigMapName: dynamic-plugins-rhdh
# ...
```

2. Save the configuration changes.

5.3. CUSTOMIZING TELEMETRY SEGMENT SOURCE

The **analytics-provider-segment** plugin sends the collected telemetry data to Red Hat by default. However, you can configure a new Segment source that receives telemetry data based on your needs. For configuration, you need a unique Segment write key that points to the Segment source.

NOTE

By configuring a new Segment source, you can collect and analyze the same set of data that is mentioned in the Telemetry data collection section. You might also require to create your own telemetry data collection notice for your application users.

Procedure

1. Configure integration with your Segment source by using one of the following options:

   Using Helm Chart

   - Add the following YAML code in your Helm configuration file:

     ```
     # ...
     upstream:
       backstage:
         extraEnvVars:
           - name: SEGMENT_WRITE_KEY
             value: <segment_key> 1
     # ...
     ```

     1 **<segment_key>** denotes a unique identifier for your Segment source.

   Using the Operator

   - Add the following YAML code in your **Backstage** custom resource (CR):

     ```
     # ...
     extraEnvs:
       envs:
         - name: SEGMENT_WRITE_KEY
           value: <segment_key> 1
     # ...
     ```

     1 **<segment_key>** denotes a unique identifier for your Segment source.

2. Save the configuration changes.

CHAPTER 6. ENABLING OBSERVABILITY FOR RED HAT DEVELOPER HUB ON OPENSHIFT CONTAINER PLATFORM

In OpenShift Container Platform, metrics are exposed through an HTTP service endpoint under the **/metrics** canonical name. You can create a **ServiceMonitor** custom resource (CR) to scrape metrics from a service endpoint in a user-defined project.

6.1. ENABLING METRICS MONITORING IN A HELM CHART INSTALLATION ON AN OPENSHIFT CONTAINER PLATFORM CLUSTER

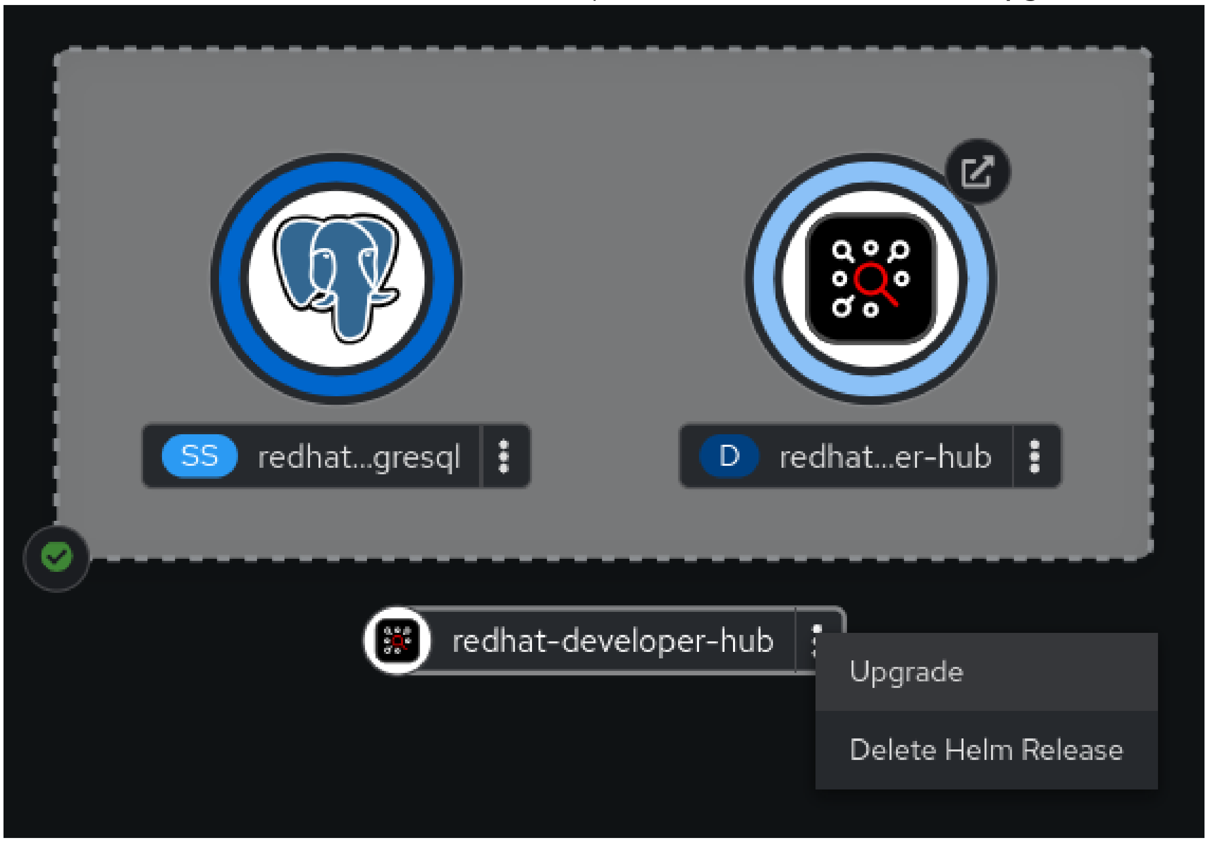
You can enable and view metrics for a Red Hat Developer Hub Helm deployment from the **Developer** perspective of the OpenShift Container Platform web console.

Prerequisites

- Your OpenShift Container Platform cluster has monitoring for user-defined projects enabled.
- You have installed Red Hat Developer Hub on OpenShift Container Platform using the Helm chart.

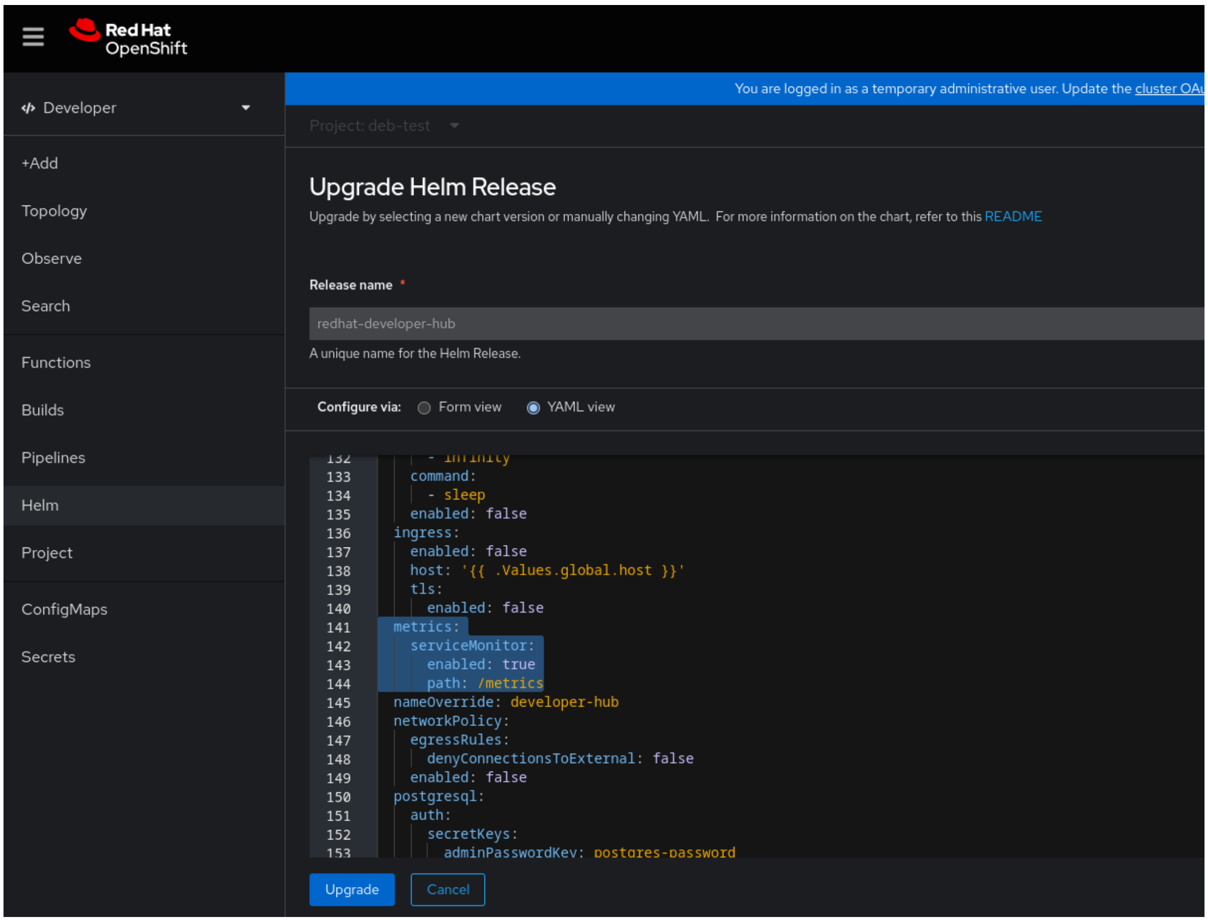
Procedure

1. From the **Developer** perspective in the OpenShift Container Platform web console, select the **Topology** view.
2. Click the overflow menu of the Red Hat Developer Hub Helm chart, and select **Upgrade**.



3. On the **Upgrade Helm Release** page, select the **YAML view** option in **Configure via**, then configure the **metrics** section in the YAML, as shown in the following example:

```
upstream:
# ...
  metrics:
    serviceMonitor:
      enabled: true
      path: /metrics
# ...
```



Red Hat OpenShift

Developer

+Add
Topology
Observe
Search
Functions
Builds
Pipelines
Helm
Project
ConfigMaps
Secrets

You are logged in as a temporary administrative user. Update the cluster OA

Project: deb-test

Upgrade Helm Release

Upgrade by selecting a new chart version or manually changing YAML. For more information on the chart, refer to this README

Release name *

redhat-developer-hub

A unique name for the Helm Release.

Configure via: Form view [✓ YAML view]

```
      - infinity
    command:
      - sleep
    enabled: false
  ingress:
    enabled: false
    host: '{{ .Values.global.host }}'
    tls:
      enabled: false
  metrics:
    serviceMonitor:
      enabled: true
      path: /metrics
  nameOverride: developer-hub
  networkPolicy:
    egressRules:
      denyConnectionsToExternal: false
    enabled: false
  postgresql:
    auth:
      secretKeys:
        adminPasswordKey: postgres-password
```

Upgrade Cancel

4. Click **Upgrade**.

Verification

1. From the **Developer** perspective in the OpenShift Container Platform web console, select the **Observe** view.

2. Click the **Metrics** tab to view metrics for Red Hat Developer Hub pods.

6.2. ENABLING METRICS MONITORING IN A RED HAT DEVELOPER HUB OPERATOR INSTALLATION ON AN OPENSHIFT CONTAINER PLATFORM CLUSTER

You can enable and view metrics for an Operator-installed Red Hat Developer Hub instance from the **Developer** perspective of the OpenShift Container Platform web console.

Prerequisites

- Your OpenShift Container Platform cluster has monitoring for user-defined projects enabled.

- You have installed Red Hat Developer Hub on OpenShift Container Platform using the Red Hat Developer Hub Operator.
- You have installed the OpenShift CLI (**oc**).

Procedure

Currently, the Red Hat Developer Hub Operator does not support creating a **ServiceMonitor** custom resource (CR) by default. You must complete the following steps to create a **ServiceMonitor** CR to scrape metrics from the endpoint.

1. Create the **ServiceMonitor** CR as a YAML file:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: <custom_resource_name> (1)
  namespace: <project_name> (2)
  labels:
    app.kubernetes.io/instance: <custom_resource_name>
    app.kubernetes.io/name: backstage
spec:
  namespaceSelector:
    matchNames:
      - <project_name>
  selector:
    matchLabels:
      rhdh.redhat.com/app: backstage-<custom_resource_name>
  endpoints:
  - port: backend
    path: '/metrics'
```

(1) Replace **<custom_resource_name>** with the name of your Red Hat Developer Hub CR.

(2) Replace **<project_name>** with the name of the OpenShift Container Platform project where your Red Hat Developer Hub instance is running.

2. Apply the **ServiceMonitor** CR by running the following command:

```
oc apply -f <filename>
```

Verification

1. From the **Developer** perspective in the OpenShift Container Platform web console, select the **Observe** view.
2. Click the **Metrics** tab to view metrics for Red Hat Developer Hub pods.

6.3. ADDITIONAL RESOURCES

- OpenShift Container Platform - Managing metrics

CHAPTER 7. RUNNING THE RHDH APPLICATION BEHIND A CORPORATE PROXY

You can run the RHDH application behind a corporate proxy by setting any of the following environment variables before starting the application:

- **HTTP_PROXY**: Denotes the proxy to use for HTTP requests.
- **HTTPS_PROXY**: Denotes the proxy to use for HTTPS requests.

Additionally, you can set the **NO_PROXY** environment variable to exclude certain domains from proxying. The variable value is a comma-separated list of hostnames that do not require a proxy to get reached, even if one is specified.

7.1. CONFIGURING PROXY INFORMATION IN HELM DEPLOYMENT

For Helm-based deployment, either a developer or a cluster administrator with permissions to create resources in the cluster can configure the proxy variables in a **values.yaml** Helm configuration file.

Prerequisites

- You have installed the Red Hat Developer Hub application.

Procedure

1. Set the proxy information in your Helm configuration file:

```
upstream:
  backstage:
    extraEnvVars:
      - name: HTTP_PROXY
        value: '<http_proxy_url>'
      - name: HTTPS_PROXY
        value: '<https_proxy_url>'
      - name: NO_PROXY
        value: '<no_proxy_settings>'
```

Where,

<http_proxy_url>
Denotes a variable that you must replace with the HTTP proxy URL.

<https_proxy_url>
Denotes a variable that you must replace with the HTTPS proxy URL.

<no_proxy_settings>
Denotes a variable that you must replace with comma-separated URLs, which you want to exclude from proxying, for example, **foo.com,baz.com**.

Example: Setting proxy variables using Helm Chart

```
upstream:
  backstage:
    extraEnvVars:
```

```
- name: HTTP_PROXY
  value: 'http://10.10.10.105:3128'
- name: HTTPS_PROXY
  value: 'http://10.10.10.106:3128'
- name: NO_PROXY
  value: 'localhost,example.org'
```

2. Save the configuration changes.

7.2. CONFIGURING PROXY INFORMATION IN OPERATOR DEPLOYMENT

For Operator-based deployment, the approach you use for proxy configuration is based on your role:

- As a cluster administrator with access to the Operator namespace, you can configure the proxy variables in the Operator's default ConfigMap file. This configuration applies the proxy settings to all the users of the Operator.
- As a developer, you can configure the proxy variables in a custom resource (CR) file. This configuration applies the proxy settings to the RHDH application created from that CR.

Prerequisites

- You have installed the Red Hat Developer Hub application.

Procedure

1. Perform one of the following steps based on your role:
   - As an administrator, set the proxy information in the Operator's default ConfigMap file:
     a. Search for a ConfigMap file named **backstage-default-config** in the default namespace **rhdh-operator** and open it.
     b. Find the **deployment.yaml** key.
     c. Set the value of the **HTTP_PROXY**, **HTTPS_PROXY**, and **NO_PROXY** environment variables in the **Deployment** spec as shown in the following example:

        Example: Setting proxy variables in a ConfigMap file

```
# Other fields omitted
  deployment.yaml: |-
    apiVersion: apps/v1
    kind: Deployment
    spec:
      template:
        spec:
          # Other fields omitted
          initContainers:
            - name: install-dynamic-plugins
              # command omitted
              env:
                - name: NPM_CONFIG_USERCONFIG
          value: /opt/app-root/src/.npmrc.dynamic-plugins
        - name: HTTP_PROXY
          value: 'http://10.10.10.105:3128'
        - name: HTTPS_PROXY
          value: 'http://10.10.10.106:3128'
        - name: NO_PROXY
          value: 'localhost,example.org'
      # Other fields omitted
    containers:
      - name: backstage-backend
      # Other fields omitted
      env:
        - name: APP_CONFIG_backend_listen_port
          value: "7007"
        - name: HTTP_PROXY
          value: 'http://10.10.10.105:3128'
        - name: HTTPS_PROXY
          value: 'http://10.10.10.106:3128'
        - name: NO_PROXY
          value: 'localhost,example.org'
```

- As a developer, set the proxy information in your custom resource (CR) file as shown in the following example:

  Example: Setting proxy variables in a CR file

  ```
  spec:
    # Other fields omitted
    application:
      extraEnvs:
        envs:
          - name: HTTP_PROXY
            value: 'http://10.10.10.105:3128'
          - name: HTTPS_PROXY
            value: 'http://10.10.10.106:3128'
          - name: NO_PROXY
            value: 'localhost,example.org'
  ```

2. Save the configuration changes.

CHAPTER 8. 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

8.1. DEPLOYING RED HAT DEVELOPER HUB ON 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 installed the **kubectl** CLI.
- You are logged into your cluster using **kubectl**, and have **developer** or **admin** permissions.
- 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 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 and activate the *<rhdh>* namespace:

```
DEPLOYMENT_NAME=<redhat-developer-hub>
NAMESPACE=<rhdh>
kubectl create namespace ${NAMESPACE}
kubectl config set-context --current --namespace=${NAMESPACE}
```

3. Create a pull secret, which is used to pull the Developer Hub images from the Red Hat Ecosystem, by running 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.

4. 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
```

5. 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 1.2.1] \
  --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

6. Configure your Developer Hub Helm chart instance with the Developer Hub database password and router base URL values from your cluster:

```
PASSWORD=$(kubectl get secret redhat-developer-hub-postgresql -o jsonpath="
{.data.password}" | base64 -d)
CLUSTER_ROUTER_BASE=$(kubectl get route console -n openshift-console -
o=jsonpath='{.spec.host}' | sed 's/^[^.]*\.//')
helm upgrade $DEPLOYMENT_NAME -i "https://github.com/openshift-helm-
charts/charts/releases/download/redhat-redhat-developer-hub-1.2.1/redhat-developer-hub-
1.2.1.tgz" \
    --set global.clusterRouterBase="$CLUSTER_ROUTER_BASE" \
    --set global.postgresql.auth.password="$PASSWORD"
```

7. Display the running Developer Hub instance URL, by running the following command:

```
echo "https://$DEPLOYMENT_NAME-$NAMESPACE.$CLUSTER_ROUTER_BASE"
```

Verification

- Open the running Developer Hub instance URL in your browser to use Developer Hub.

Upgrade

- To upgrade the deployment, run the following command:

```
helm upgrade $DEPLOYMENT_NAME -i https://github.com/openshift-helm-
charts/charts/releases/download/redhat-redhat-developer-hub-1.2.1/redhat-developer-hub-
1.2.1.tgz
```

Delete

- To delete the deployment, run the following command:

```
helm -n $NAMESPACE delete $DEPLOYMENT_NAME
```

8.2. 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.

8.2.1. 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 **kubectl**. For more information, see Installing or updating kubectl.
- 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

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

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

2. 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
```

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 **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
```

4. 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
```

5. 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.v1.2.0
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.

8.2.2. 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 **rhdh-operator-<VERSION>.yaml**, which is updated manually.

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

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

4. Open the generated deployment manifest file in an editor and perform the following steps:

a. 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 ---
```

b. 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 ---
```

c. 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 ---
```

d. 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-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:
        kubectl.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
```

8.2.3. 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 **kubeconfig**. For more information, see Creating or updating a kubeconfig file for an Amazon {eks} cluster .
- You have installed **kubectl**. For more information, see Installing or updating kubectl .

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")'
```

3. 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>
```

4. 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"
```

5. 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.

8.3. 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.

8.3.1. 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.

8.3.1.1. 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**.

8.3.2. 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.

8.3.2.1. 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
```

8.3.2.2. 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"]
```

8.4. 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 fedarate 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:

  1. 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**.
  2. 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**.
  3. Under **OAuth 2.0 grant types**, select **Authorization code grant** to return an authorization code.
  4. 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 1.2.1] \
--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 9. 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

9.1. DEPLOYING RED HAT DEVELOPER HUB ON AZURE KUBERNETES SERVICE (AKS) USING THE HELM CHART

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

Prerequisites

- You have an Azure account with active subscription.
- You have installed the Azure CLI.
- You have installed the **kubectl** CLI.
- You are logged into your cluster using **kubectl**, and have **developer** or **admin** permissions.
- 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 addresssed 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 command:

  ```
  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>
  ```

Procedure

1. Log in to AKS by running the following command:

   ```
   az login [--tenant=<optional_directory_name>]
   ```

2. Create a resource group by running the following command:

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

   TIP

   You can list available regions by running the following command:

   ```
   az account list-locations -o table
   ```

3. Create an AKS cluster by running the following command:

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

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

4. Connect to your cluster by running the following command:

   ```
   az aks get-credentials --resource-group <resource_group_name> --name <cluster_name>
   ```

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

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

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

6. Create and activate the *<rhdh>* namespace:

```
DEPLOYMENT_NAME=<redhat-developer-hub>
NAMESPACE=<rhdh>
kubectl create namespace ${NAMESPACE}
kubectl config set-context --current --namespace=${NAMESPACE}
```

7. Create a pull secret, which is used to pull the Developer Hub images from the Red Hat Ecosystem, by running the following command:

```
kubectl -n $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>
```

8. 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
```

9. To install Developer Hub by using the Helm chart, run the following command:

```
helm -n $NAMESPACE install -f values.yaml $DEPLOYMENT_NAME openshift-helm-charts/redhat-developer-hub --version 1.2.1
```

10. Verify the deployment status:

```
kubectl get deploy $DEPLOYMENT_NAME -n $NAMESPACE
```

11. Configure your Developer Hub Helm chart instance with the Developer Hub database password and router base URL values from your cluster:

    ```
    PASSWORD=$(kubectl get secret redhat-developer-hub-postgresql -o jsonpath="
    {.data.password}" | base64 -d)
    CLUSTER_ROUTER_BASE=$(kubectl get route console -n openshift-console -
    o=jsonpath='{.spec.host}' | sed 's/^[^.]*\.//')
    helm upgrade $DEPLOYMENT_NAME -i "https://github.com/openshift-helm-
    charts/charts/releases/download/redhat-redhat-developer-hub-1.2.1/redhat-developer-hub-
    1.2.1.tgz" \
        --set global.clusterRouterBase="$CLUSTER_ROUTER_BASE" \
        --set global.postgresql.auth.password="$PASSWORD"
    ```

12. Display the running Developer Hub instance URL, by running the following command:

    ```
    echo "https://$DEPLOYMENT_NAME-$NAMESPACE.$CLUSTER_ROUTER_BASE"
    ```

Verification

- Open the running Developer Hub instance URL in your browser to use Developer Hub.

Upgrade

- To upgrade the deployment, run the following command:

  ```
  helm upgrade $DEPLOYMENT_NAME -i https://github.com/openshift-helm-
  charts/charts/releases/download/redhat-redhat-developer-hub-1.2.1/redhat-developer-hub-
  1.2.1.tgz
  ```

Delete

- To delete the deployment, run the following command:

  ```
  helm -n $NAMESPACE delete $DEPLOYMENT_NAME
  ```

9.2. 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.

Prerequisites

- You have an Azure account with active subscription.
- You have installed the Azure CLI.
- You have installed the **kubectl** CLI.
- You are logged into your cluster using **kubectl**, and have **developer** or **admin** permissions.
- 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 addresssed 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 command:

  ```
  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>
  ```

Procedure

1. Log in to AKS by running the following command:

   ```
   az login [--tenant=<optional_directory_name>]
   ```

2. Create a resource group by running the following command:

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

   TIP

   You can list available regions by running the following command:

   ```
   az account list-locations -o table
   ```

3. Create an AKS cluster by running the following command:

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

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

4. Connect to your cluster by running the following command:

   ```
   az aks get-credentials --resource-group <resource_group_name> --name <cluster_name>
   ```

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

5. 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
   ```

6. 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.

7. 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>
   ```

8. 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
```

9. To deploy the created Ingress, run the following command:

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

10. 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>
```

11. 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"
```

12. 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"
```

13. Apply the CR manifest to your namespace:

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

14. 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**).

15. Optional: To delete the CR, run the following command:

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

9.3. 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.

9.3.1. 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 9.1, "Deploying Red Hat Developer Hub on Azure Kubernetes Service (AKS) using the Helm chart".

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.

9.4. 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 9.1, "Deploying Red Hat Developer Hub on Azure Kubernetes Service (AKS) using the Helm chart".
- You have created registered your application in Azure portal. For more information, see Register an application with the Microsoft identity platform.

9.4.1. 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 9.1, "Deploying 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 1.2.1
```

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>
```

9.4.2. 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 10. 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.

10.1. 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.

You can define the following types of permissions in Developer Hub:

- resource type
- basic

The distinction between the two permission types depend on whether a permission includes a defined resource type.

You can define the resource type permission using either the associated resource type or the permission name as shown in the following example:

Example resource type permission definition

```
p, role:default/myrole, catalog.entity.read, read, allow
g, user:default/myuser, role:default/myrole

p, role:default/another-role, catalog-entity, read, allow
g, user:default/another-user, role:default/another-role
```

You can define the basic permission in Developer Hub using the permission name as shown in the following example:

Example basic permission definition

```
p, role:default/myrole, catalog.entity.create, create, allow
g, user:default/myuser, role:default/myrole
```

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
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
scaffolder.task.create		create	Allows the user or role to trigger software templates which create new scaffolder tasks
scaffolder.task.cancel			Allows the user or role to cancel currently running scaffolder tasks

Name	Resource type	Policy	Description
scaffolder.task.read		read	Allows user or role to read all scaffolder tasks and their associated events and logs

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

OCM permissions

Name	Resource type	Policy	Description
ocm.entity.read		read	Allows user or role to read from the OCM plugin
ocm.cluster.read		read	Allows user or role to read the cluster information in the OCM plugin

Topology permissions

Name	Resource type	Policy	Description
topology.view.read		read	Allows user or role to view the topology plugin

Name	Resource type	Policy	Description
kubernetes .proxy			Allows user or role to access the proxy endpoint, allowing them to read pod logs and events within RHDH

10.1.1. 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

10.1.1.1. 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
```

The permission policy role (**role:default/rbac_admin**) is a default role in Developer Hub and includes some permissions upon creation, such as creating, reading, updating, and deleting permission policies/roles, as well as reading from the catalog.

If the default permissions are not adequate for your requirements, you can define a new administrator role tailored to your requirements using relevant permission policies. Alternatively, you can use the optional **superUsers** configuration value, which grants unrestricted permissions across Developer Hub.

You can set the **superUsers** in the **app-config.yaml** file as follows:

```
# ...
permission:
  enabled: true
  rbac:
    admin:
```

```
superUsers:
  - name: user:default/joeuser
  # ...
```

10.1.1.2. Configuration of permission policies defined in an external file

You can configure 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. You must define the permission policies using the following Casbin rules format:

```
---
`p, <ROLE>, <PERMISSION_NAME or PERMISSION_RESOURCE_TYPE>,
<PERMISSION_POLICY_ACTION>, <ALLOW or DENY>`
---
```

You can define roles using the following Casbin rules format:

```
---
`g, <USER or GROUP>, <ROLE>`
---
```

NOTE

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, allow`
```

p, role:default/guests, catalog.entity.create, create, allow

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

g, group:default/<GROUP_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, allow`
---
```

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
```

You can use an optional configuration value that enables reloading the CSV file without restarting the Developer Hub instance.

Set the value of the **policyFileReload** option in the **app-config.yaml** file:

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

10.1.1.2.1. Mounting **policy.csv** file to the Developer Hub Helm chart

When the Red Hat Developer Hub is deployed with the Helm chart, 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
    ```

10.2. CONDITIONAL POLICIES IN RED HAT DEVELOPER HUB

The permission framework in Red Hat Developer Hub provides conditions, supported by the RBAC backend plugin (**backstage-plugin-rbac-backend**). The conditions work as content filters for the Developer Hub resources that are provided by the RBAC backend plugin.

The RBAC backend API stores conditions assigned to roles in the database. When you request to access the frontend resources, the RBAC backend API searches for the corresponding conditions and delegates them to the appropriate plugin using its plugin ID. If you are assigned to multiple roles with different conditions, then the RBAC backend merges the conditions using the **anyOf** criteria.

Conditional criteria

A condition in Developer Hub is a simple condition with a rule and parameters. However, a condition can also contain a parameter or an array of parameters combined by conditional criteria. The supported conditional criteria includes:

- **allOf**: Ensures that all conditions within the array must be true for the combined condition to be satisfied.
- **anyOf**: Ensures that at least one of the conditions within the array must be true for the combined condition to be satisfied.
- **not**: Ensures that the condition within it must not be true for the combined condition to be satisfied.

Conditional object

The plugin specifies the parameters supported for conditions. You can access the conditional object schema from the RBAC API endpoint to understand how to construct a conditional JSON object, which is then used by the RBAC backend plugin API.
A conditional object contains the following parameters:

Table 10.1. Conditional object parameters

Parameter	Type	Description
result	String	Always has the value **CONDITIONAL**
roleEntityRef	String	String entity reference to the RBAC role, such as **role:default/dev**
pluginId	String	Corresponding plugin ID, such as **catalog**
permissionMapping	String array	Array permission actions, such as **['read', 'update', 'delete']**
resourceType	String	Resource type provided by the plugin, such as **catalog-entity**
conditions	JSON	Condition JSON with parameters or array parameters joined by criteria

10.2.1. Conditional policies definition

You can access API endpoints for conditional policies in Red Hat Developer Hub. For example, to retrieve the available conditional rules, which can help you define these policies, you can access the **GET [api/plugins/condition-rules]** endpoint.

The **api/plugins/condition-rules** returns the condition parameters schemas, for example:

```
[
  {
    "pluginId": "catalog",
    "rules": [
      {
        "name": "HAS_ANNOTATION",
        "description": "Allow entities with the specified annotation",
        "resourceType": "catalog-entity",
        "paramsSchema": {
          "type": "object",
          "properties": {
            "annotation": {
              "type": "string",
              "description": "Name of the annotation to match on"
            },
            "value": {
              "type": "string",
          "description": "Value of the annotation to match on"
        }
      },
      "required": [
        "annotation"
      ],
      "additionalProperties": false,
      "$schema": "http://json-schema.org/draft-07/schema#"
    }
  },
  {
    "name": "HAS_LABEL",
    "description": "Allow entities with the specified label",
    "resourceType": "catalog-entity",
    "paramsSchema": {
      "type": "object",
      "properties": {
        "label": {
          "type": "string",
          "description": "Name of the label to match on"
        }
      },
      "required": [
        "label"
      ],
      "additionalProperties": false,
      "$schema": "http://json-schema.org/draft-07/schema#"
    }
  },
  {
    "name": "HAS_METADATA",
    "description": "Allow entities with the specified metadata subfield",
    "resourceType": "catalog-entity",
    "paramsSchema": {
      "type": "object",
      "properties": {
        "key": {
          "type": "string",
          "description": "Property within the entities metadata to match on"
        },
        "value": {
          "type": "string",
          "description": "Value of the given property to match on"
        }
      },
      "required": [
        "key"
      ],
      "additionalProperties": false,
      "$schema": "http://json-schema.org/draft-07/schema#"
    }
  },
  {
    "name": "HAS_SPEC",
    "description": "Allow entities with the specified spec subfield",
    "resourceType": "catalog-entity",
      "paramsSchema": {
        "type": "object",
        "properties": {
          "key": {
            "type": "string",
            "description": "Property within the entities spec to match on"
          },
          "value": {
            "type": "string",
            "description": "Value of the given property to match on"
          }
        },
        "required": [
          "key"
        ],
        "additionalProperties": false,
        "$schema": "http://json-schema.org/draft-07/schema#"
      }
    },
    {
      "name": "IS_ENTITY_KIND",
      "description": "Allow entities matching a specified kind",
      "resourceType": "catalog-entity",
      "paramsSchema": {
        "type": "object",
        "properties": {
          "kinds": {
            "type": "array",
            "items": {
              "type": "string"
            },
            "description": "List of kinds to match at least one of"
          }
        },
        "required": [
          "kinds"
        ],
        "additionalProperties": false,
        "$schema": "http://json-schema.org/draft-07/schema#"
      }
    },
    {
      "name": "IS_ENTITY_OWNER",
      "description": "Allow entities owned by a specified claim",
      "resourceType": "catalog-entity",
      "paramsSchema": {
        "type": "object",
        "properties": {
          "claims": {
            "type": "array",
            "items": {
              "type": "string"
            },
            "description": "List of claims to match at least one on within ownedBy"
          }
        },
            "required": [
              "claims"
            ],
            "additionalProperties": false,
            "$schema": "http://json-schema.org/draft-07/schema#"
          }
        }
      ]
    }
    ... <another plugin condition parameter schemas>
]
```

The RBAC backend API constructs a condition JSON object based on the previous condition schema.

10.2.1.1. Examples of conditional policies

In Red Hat Developer Hub, you can define conditional policies with or without criteria. You can use the following examples to define the conditions based on your use case:

A condition without criteria

Consider a condition without criteria displaying catalogs only if user is a member of the owner group. To add this condition, you can use the catalog plugin schema **IS_ENTITY_OWNER** as follows:

Example condition without criteria

```
{
  "rule": "IS_ENTITY_OWNER",
  "resourceType": "catalog-entity",
  "params": {
    "claims": ["group:default/team-a"]
  }
}
```

In the previous example, the only conditional parameter used is **claims**, which contains a list of user or group entity references.

You can apply the previous example condition to the RBAC REST API by adding additional parameters as follows:

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/test",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["read"],
  "conditions": {
    "rule": "IS_ENTITY_OWNER",
    "resourceType": "catalog-entity",
    "params": {
      "claims": ["group:default/team-a"]
    }
  }
}
```

A condition with criteria

Consider a condition with criteria, which displays catalogs only if user is a member of owner group OR displays list of all catalog user groups.
To add the criteria, you can add another rule as **IS_ENTITY_KIND** in the condition as follows:

Example condition with criteria

```
{
  "anyOf": [
    {
      "rule": "IS_ENTITY_OWNER",
      "resourceType": "catalog-entity",
      "params": {
        "claims": ["group:default/team-a"]
      }
    },
    {
      "rule": "IS_ENTITY_KIND",
      "resourceType": "catalog-entity",
      "params": {
        "kinds": ["Group"]
      }
    }
  ]
}
```

NOTE

Running conditions in parallel during creation is not supported. Therefore, consider defining nested conditional policies based on the available criteria.

Example of nested conditions

```
{
  "anyOf": [
    {
      "rule": "IS_ENTITY_OWNER",
      "resourceType": "catalog-entity",
      "params": {
        "claims": ["group:default/team-a"]
      }
    },
    {
      "rule": "IS_ENTITY_KIND",
      "resourceType": "catalog-entity",
      "params": {
        "kinds": ["Group"]
      }
    }
  ],
  "not": {
    "rule": "IS_ENTITY_KIND",
    "resourceType": "catalog-entity",
    "params": { "kinds": ["Api"] }
  }
}
```

You can apply the previous example condition to the RBAC REST API by adding additional parameters as follows:

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/test",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["read"],
  "conditions": {
    "anyOf": [
      {
        "rule": "IS_ENTITY_OWNER",
        "resourceType": "catalog-entity",
        "params": {
          "claims": ["group:default/team-a"]
        }
      },
      {
        "rule": "IS_ENTITY_KIND",
        "resourceType": "catalog-entity",
        "params": {
          "kinds": ["Group"]
        }
      }
    ]
  }
}
```

The following examples can be used with Developer Hub plugins. These examples can help you determine how to define conditional policies:

Conditional policy defined for Keycloak plugin

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/developer",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["update", "delete"],
  "conditions": {
    "not": {
      "rule": "HAS_ANNOTATION",
      "resourceType": "catalog-entity",
      "params": { "annotation": "keycloak.org/realm", "value": "<YOUR_REALM>" }
    }
  }
}
```

[illegible]

The previous example of Keycloak plugin prevents users in the **role:default/developer** from updating or deleting users that are ingested into the catalog from the Keycloak plugin.

NOTE

In the previous example, the annotation **keycloak.org/realm** requires the value of **<YOUR_REALM>**.

Conditional policy defined for Quay plugin

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/developer",
  "pluginId": "scaffolder",
  "resourceType": "scaffolder-action",
  "permissionMapping": ["use"],
  "conditions": {
    "not": {
      "rule": "HAS_ACTION_ID",
      "resourceType": "scaffolder-action",
      "params": { "actionId": "quay:create-repository" }
    }
  }
}
```

The previous example of Quay plugin prevents the role **role:default/developer** from using the Quay scaffolder action. Note that **permissionMapping** contains **use**, signifying that **scaffolder-action** resource type permission does not have a permission policy.

For more information about permissions in Red Hat Developer Hub, see Section 10.1, “Permission policies in Red Hat Developer Hub”.

10.3. 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 Configuring plugins in Red Hat Developer Hub .

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.

10.3.1. 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 Configuring plugins in Red Hat Developer Hub .
- You have configured the required permission policies. For more information, see Section 10.1.1, "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.

10.3.2. 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 Configuring plugins in Red Hat Developer Hub .
- You have configured the required permission policies. For more information, see Section 10.1.1, "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.

10.3.3. 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 Configuring plugins in Red Hat Developer Hub .
- You have configured the required permission policies. For more information, see Section 10.1.1, "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**.

10.4. 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 Developer Hub **Homepage**, navigate to the **Network** tab and search for the **query?term=** network call. Alternatively, you can go to the **Catalog** page and select any Catalog API network call 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 **name**, 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

Source

Each permission policy and role created using the RBAC plugin is associated with a source to maintain data consistency within the plugin. You can manipulate permission policies and roles based on the following designated source information:

- CSV file
- Configuration file
- REST API
- Legacy

Managing roles and permission policies originating from CSV files and REST API involves straightforward modification based on their initial source information.

The Configuration file pertains to the default **role:default/rbac_admin** role provided by the RBAC plugin. The default role has limited permissions to create, read, update, and delete permission policies or roles, and to read catalog entities.

NOTE

In case the default permissions are insufficient for your administrative requirements, you can create a custom admin role with required permission policies.

The legacy source applies to policies and roles defined before RBAC backend plugin version **2.1.3**, and is the least restrictive among the source location options. You must update the permissions and roles in legacy source to use either REST API or the CSV file sources.

You can use the **GET** requests to query roles and policies and determine the source information, if required.

10.4.1. 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.

10.4.2. Supported RBAC REST API endpoints

The RBAC REST API provides endpoints for managing roles, permissions, and conditional policies in the Developer Hub and for retrieving information about the roles and policies.

10.4.2.1. 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/username"],
    "name": "role:default/guests"
  },
  {
    "memberReferences": [
      "group:default/groupname",
      "user:default/username"
    ],
    "name": "role:default/rbac_admin"
  }
]
```

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

Returns information for a single role in Developer Hub.

Example response (JSON)

```
[
  {
    "memberReferences": [
      "group:default/groupname",
      "user:default/username"
    ],
    "name": "role:default/rbac_admin"
  }
]
```

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

Creates a role in Developer Hub.

Table 10.2. Request parameters

Name	Description	Type	Presence
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	Presence
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 10.3. Request parameters

Name	Description	Type	Presence
kind	Kind of the entity	String	Required
namespace	Namespace of the entity	String	Required
name	Name of 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 10.4. Request parameters

Name	Description	Type	Presence
kind	Kind of the entity	String	Required
namespace	Namespace of the entity	String	Required
name	Name of the entity	String	Required

Example response

```
204
```

10.4.2.2. 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",
    "metadata": {
      "source": "csv-file"
    }
  },
  {
    "entityReference": "role:default/test",
    "permission": "catalog.entity.create",
    "policy": "use",
    "effect": "allow",
    "metadata": {
      "source": "csv-file"
    }
  },
]
```

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

Returns permission policies related to the specified entity reference.

Table 10.5. Request parameters

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

Example response (JSON)

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

[POST] /api/permission/policies

Creates a permission policy for a specified entity.

Table 10.6. Request parameters

Name	Description	Type	Presence
entityReference	Reference values of an entity including **kind**, **namespace**, and **name**	String	Required
permission	Permission from a specific plugin, resource type, or name	String	Required
policy	Policy action for the permission, such as **create**, **read**, **update**, **delete**, or **use**	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	Presence
permission	Permission from a specific plugin, resource type, or name	String	Required
policy	Policy action for the permission, such as **create**, **read**, **update**, **delete**, or **use**	String	Required
effect	Indication of allowing or not allowing the policy	String	Required

Example request body (JSON)

```
{
  "oldPolicy": [
    {
      "permission": "catalog-entity",
      "policy": "read",
      "effect": "allow"
    },
    {
      "permission": "catalog.entity.create",
      "policy": "create",
      "effect": "allow"
    }
  ],
  "newPolicy": [
    {
      "permission": "catalog-entity",
      "policy": "read",
      "effect": "deny"
    },
    {
      "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 10.7. Request parameters

Name	Description	Type	Presence
kind	Kind of the entity	String	Required
namespace	Namespace of the entity	String	Required
name	Name related to the entity	String	Required
permission	Permission from a specific plugin, resource type, or name	String	Required
policy	Policy action for the permission, such as **create**, **read**, **update**, **delete**, or **use**	String	Required
effect	Indication of allowing or not allowing the policy	String	Required

Example response

```
204 No Content
```

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

Deletes all permission policies added to the specified entity.

Table 10.8. Request parameters

Name	Description	Type	Presence
kind	Kind of the entity	String	Required
namespace	Namespace of the entity	String	Required
name	Name related to the entity	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": [
      {
        "isResourced": true,
        "permission": "catalog-entity",
        "policy": "read"
      },
      {
        "isResourced": false,
        "permission": "catalog.entity.create",
        "policy": "create"
      },
      {
        "isResourced": true,
        "permission": "catalog-entity",
        "policy": "delete"
      },
      {
        "isResourced": true,
        "permission": "catalog-entity",
        "policy": "update"
      },
      {
        "isResourced": false,
        "permission": "catalog.location.read",
        "policy": "read"
      },
      {
        "isResourced": false,
        "permission": "catalog.location.create",
        "policy": "create"
      },
      {
        "isResourced": false,
        "permission": "catalog.location.delete",
        "policy": "delete"
      }
    ]
  },
  ...
]
```

10.4.2.3. Conditional policies

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

[GET] /api/plugins/condition-rules

Returns available conditional rule parameter schemas for the available plugins that are enabled in Developer Hub.

Example response (JSON)

```
[
{
  "pluginId": "catalog",
  "rules": [
    {
      "name": "HAS_ANNOTATION",
      "description": "Allow entities with the specified annotation",
      "resourceType": "catalog-entity",
      "paramsSchema": {
        "type": "object",
        "properties": {
          "annotation": {
            "type": "string",
            "description": "Name of the annotation to match on"
          },
          "value": {
            "type": "string",
            "description": "Value of the annotation to match on"
          }
        },
        "required": [
          "annotation"
        ],
        "additionalProperties": false,
        "$schema": "http://json-schema.org/draft-07/schema#"
      }
    },
    {
      "name": "HAS_LABEL",
      "description": "Allow entities with the specified label",
      "resourceType": "catalog-entity",
      "paramsSchema": {
        "type": "object",
        "properties": {
          "label": {
            "type": "string",
            "description": "Name of the label to match on"
          }
        },
        "required": [
          "label"
        ],
        "additionalProperties": false,
        "$schema": "http://json-schema.org/draft-07/schema#"
      }
    },
    {
      "name": "HAS_METADATA",
      "description": "Allow entities with the specified metadata subfield",
      "resourceType": "catalog-entity",
      "paramsSchema": {
        "type": "object",
        "properties": {
          "key": {
            "type": "string",
            "description": "Property within the entities metadata to match on"
          },
          "value": {
            "type": "string",
            "description": "Value of the given property to match on"
          }
        },
        "required": [
          "key"
        ],
        "additionalProperties": false,
        "$schema": "http://json-schema.org/draft-07/schema#"
      }
    },
    {
      "name": "HAS_SPEC",
      "description": "Allow entities with the specified spec subfield",
      "resourceType": "catalog-entity",
      "paramsSchema": {
        "type": "object",
        "properties": {
          "key": {
            "type": "string",
            "description": "Property within the entities spec to match on"
          },
          "value": {
            "type": "string",
            "description": "Value of the given property to match on"
          }
        },
        "required": [
          "key"
        ],
        "additionalProperties": false,
        "$schema": "http://json-schema.org/draft-07/schema#"
      }
    },
    {
      "name": "IS_ENTITY_KIND",
      "description": "Allow entities matching a specified kind",
      "resourceType": "catalog-entity",
      "paramsSchema": {
        "type": "object",
        "properties": {
          "kinds": {
            "type": "array",
            "items": {
              "type": "string"
            },
            "description": "List of kinds to match at least one of"
          }
        },
        "required": [
          "kinds"
        ],
        "additionalProperties": false,
        "$schema": "http://json-schema.org/draft-07/schema#"
      }
```

```
          },
          {
            "name": "IS_ENTITY_OWNER",
            "description": "Allow entities owned by a specified claim",
            "resourceType": "catalog-entity",
            "paramsSchema": {
              "type": "object",
              "properties": {
                "claims": {
                  "type": "array",
                  "items": {
                    "type": "string"
                  },
                  "description": "List of claims to match at least one on within ownedBy"
                }
              },
              "required": [
                "claims"
              ],
              "additionalProperties": false,
              "$schema": "http://json-schema.org/draft-07/schema#"
            }
          }
        ]
      }
      ... <another plugin condition parameter schemas>
    ]
```

[GET] /api/permission/roles/conditions/:id

Returns conditions for the specified ID.

Example response (JSON)

```
{
  "id": 1,
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/test",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["read"],
  "conditions": {
    "anyOf": [
      {
        "rule": "IS_ENTITY_OWNER",
        "resourceType": "catalog-entity",
        "params": {
          "claims": ["group:default/team-a"]
        }
      },
      {
        "rule": "IS_ENTITY_KIND",
        "resourceType": "catalog-entity",
        "params": {
          "kinds": ["Group"]
        }
        }
      ]
    }
  }
```

[GET] /api/permission/roles/conditions

Returns list of all conditions for all roles.

Example response (JSON)

```
[
  {
    "id": 1,
    "result": "CONDITIONAL",
    "roleEntityRef": "role:default/test",
    "pluginId": "catalog",
    "resourceType": "catalog-entity",
    "permissionMapping": ["read"],
    "conditions": {
      "anyOf": [
        {
          "rule": "IS_ENTITY_OWNER",
          "resourceType": "catalog-entity",
          "params": {
            "claims": ["group:default/team-a"]
          }
        },
        {
          "rule": "IS_ENTITY_KIND",
          "resourceType": "catalog-entity",
          "params": {
            "kinds": ["Group"]
          }
        }
      ]
    }
  }
]
```

[POST] /api/permission/roles/conditions

Creates a conditional policy for the specified role.

Table 10.9. Request parameters

Name	Description	Type	Presence
result	Always has the value **CONDITIONAL**	String	Required
roleEntity Ref	String entity reference to the RBAC role, such as **role:default/dev**	String	Required
pluginId	Corresponding plugin ID, such as **catalog**	String	Required

Name	Description	Type	Presence
permissionMapping	Array permission action, such as **['read', 'update', 'delete']**	String array	Required
resourceType	Resource type provided by the plugin, such as **catalog-entity**	String	Required
conditions	Condition JSON with parameters or array parameters joined by criteria	JSON	Required
name	Name of the role	String	Required
metadata.description	The description of the role	String	Optional

Example request body (JSON)

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/test",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["read"],
  "conditions": {
    "rule": "IS_ENTITY_OWNER",
    "resourceType": "catalog-entity",
    "params": {
      "claims": ["group:default/team-a"]
    }
  }
}
```

Example response (JSON)

```
{
  "id": 1
}
```

[PUT] /permission/roles/conditions/:id

Updates a condition policy for a specified ID.

Table 10.10. Request parameters

Name	Description	Type	Presence
result	Always has the value **CONDITIONAL**	String	Required

Name	Description	Type	Presence
roleEntityRef	String entity reference to the RBAC role, such as **role:default/dev**	String	Required
pluginId	Corresponding plugin ID, such as **catalog**	String	Required
permissionMapping	Array permission action, such as **['read', 'update', 'delete']**	String array	Required
resourceType	Resource type provided by the plugin, such as **catalog-entity**	String	Required
conditions	Condition JSON with parameters or array parameters joined by criteria	JSON	Required
name	Name of the role	String	Required
metadata.description	The description of the role	String	Optional

Example request body (JSON)

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/test",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["read"],
  "conditions": {
    "anyOf": [
      {
        "rule": "IS_ENTITY_OWNER",
        "resourceType": "catalog-entity",
        "params": {
          "claims": ["group:default/team-a"]
        }
      },
      {
        "rule": "IS_ENTITY_KIND",
        "resourceType": "catalog-entity",
        "params": {
          "kinds": ["Group"]
        }
      }
    ]
  }
}
```

Example response

```
200
```

[DELETE] /api/permission/roles/conditions/:id

Deletes a conditional policy for the specified ID.

Example response

```
204
```

CHAPTER 11. 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.

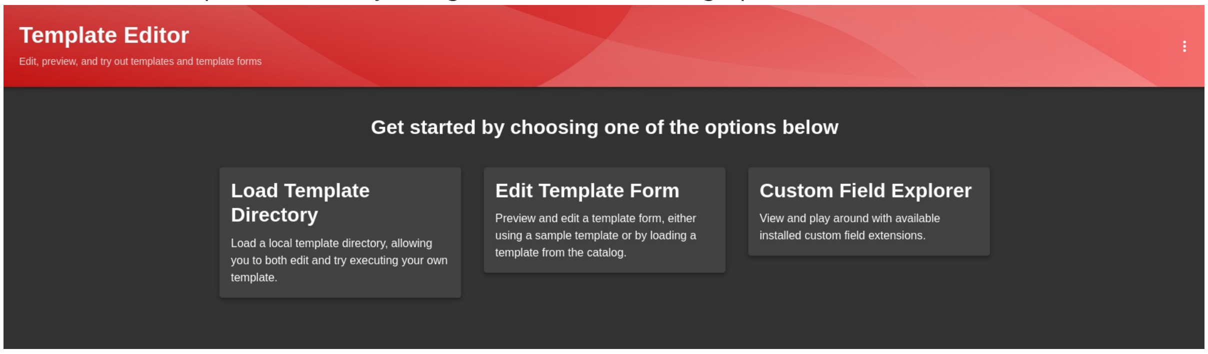
You can use templates to easily create Red Hat Developer Hub components, and then publish these components to different locations, such as the Red Hat Developer Hub software catalog, or repositories in GitHub or GitLab.

11.1. 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:



Template Editor
Edit, preview, and try out templates and template forms
Get started by choosing one of the options below
Load Template Directory
Load a local template directory, allowing you to both edit and try executing your own template.
Edit Template Form
Preview and edit a template form, either using a sample template or by loading a template from the catalog.
Custom Field Explorer
View and play around with available installed custom field extensions.

- 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 11.2, “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.

11.2. 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

11.3. 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

CHAPTER 12. CONFIGURING THE TECHDOCS PLUGIN IN RED HAT DEVELOPER HUB

The Red Hat Developer Hub TechDocs plugin helps your organization create, find, and use documentation in a central location and in a standardized way. For example:

Docs-like-code approach

Write your technical documentation in Markdown files that are stored inside your project repository along with your code.

Documentation site generation

Use MkDocs to create a full-featured, Markdown-based, static HTML site for your documentation that is rendered centrally in Developer Hub.

Documentation site metadata and integrations

See additional metadata about the documentation site alongside the static documentation, such as the date of the last update, the site owner, top contributors, open GitHub issues, Slack support channels, and Stack Overflow Enterprise tags.

Built-in navigation and search

Find the information that you want from a document more quickly and easily.

Add-ons

Customize your TechDocs experience with Add-ons to address higher-order documentation needs.

The TechDocs plugin is preinstalled and enabled on a Developer Hub instance by default. You can disable or enable the TechDocs plugin, and change other parameters, by configuring the Red Hat Developer Hub Helm chart or the Red Hat Developer Hub Operator config map.

IMPORTANT

Red Hat Developer Hub includes a built-in TechDocs builder that generates static HTML documentation from your codebase. However, the default basic setup of the local builder is not intended for production.

You can use a CI/CD pipeline with the repository that has a dedicated job to generate docs for TechDocs. The generated static files are stored in OpenShift Data Foundation or in a cloud storage solution of your choice and published to a static HTML documentation site.

After you configure OpenShift Data Foundation to store the files that TechDocs generates, you can configure the TechDocs plugin to use the OpenShift Data Foundation for cloud storage.

Additional resources

- For more information, see Configuring plugins in Red Hat Developer Hub .

12.1. CONFIGURING STORAGE FOR TECHDOCS FILES

The TechDocs publisher stores generated files in local storage or in cloud storage, such as OpenShift Data Foundation, Google GCS, AWS S3, or Azure Blob Storage.

12.1.1. Using OpenShift Data Foundation for file storage

You can configure OpenShift Data Foundation to store the files that TechDocs generates instead of relying on other cloud storage solutions.

OpenShift Data Foundation provides an **ObjectBucketClaim** custom resource (CR) that you can use to request an S3 compatible bucket backend. You must install the OpenShift Data Foundation Operator to use this feature.

Prerequisites

- An OpenShift Container Platform administrator has installed the OpenShift Data Foundation Operator in Red Hat OpenShift Container Platform. For more information, see OpenShift Container Platform - Installing Red Hat OpenShift Data Foundation Operator.
- An OpenShift Container Platform administrator has created an OpenShift Data Foundation cluster and configured the **StorageSystem** schema. For more information, see OpenShift Container Platform - Creating an OpenShift Data Foundation cluster.

Procedure

- Create an **ObjectBucketClaim** CR where the generated TechDocs files are stored. For example:

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: <rhdh_bucket_claim_name>
spec:
  generateBucketName: <rhdh_bucket_claim_name>
  storageClassName: openshift-storage.noobaa.io
```

> **NOTE**
>
> Creating the Developer Hub **ObjectBucketClaim** CR automatically creates both the Developer Hub **ObjectBucketClaim** config map and secret. The config map and secret have the same name as the **ObjetBucketClaim** CR.

After you create the **ObjectBucketClaim** CR, you can use the information stored in the config map and secret to make the information accessible to the Developer Hub container as environment variables. Depending on the method that you used to install Developer Hub, you add the access information to either the Red Hat Developer Hub Helm chart or Operator configuration.

Additional resources

- For more information about the Object Bucket Claim, see OpenShift Container Platform - Object Bucket Claim.

12.1.2. Making object storage accessible to containers by using the Helm chart

Creating a **ObjectBucketClaim** custom resource (CR) automatically generates both the Developer Hub **ObjectBucketClaim** config map and secret. The config map and secret contain **ObjectBucket** access information. Adding the access information to the Helm chart configuration makes it accessible to the Developer Hub container by adding the following environment variables to the container:

- **BUCKET_NAME**

- **BUCKET_HOST**
- **BUCKET_PORT**
- **BUCKET_REGION**
- **BUCKET_SUBREGION**
- **AWS_ACCESS_KEY_ID**
- **AWS_SECRET_ACCESS_KEY**

These variables are then used in the TechDocs plugin configuration.

Prerequisites

- You have installed Red Hat Developer Hub on OpenShift Container Platform using the Helm chart.
- You have created an **ObjectBucketClaim** CR for storing files generated by TechDocs. For more information see Using OpenShift Data Foundation for file storage

Procedure

- In the **upstream.backstage** key in the Helm chart values, enter the name of the Developer Hub **ObjectBucketClaim** secret as the value for the **extraEnvVarsSecrets** field and the **extraEnvVarsCM** field. For example:

```
upstream:
  backstage:
    extraEnvVarsSecrets:
      - <rhdh_bucket_claim_name>
    extraEnvVarsCM:
      - <rhdh_bucket_claim_name>
```

12.1.2.1. Example TechDocs Plugin configuration for the Helm chart

The following example shows a Developer Hub Helm chart configuration for the TechDocs plugin:

```
global:
  dynamic:
    includes:
      - 'dynamic-plugins.default.yaml'
  plugins:
    - disabled: false
      package: ./dynamic-plugins/dist/backstage-plugin-techdocs-backend-dynamic
      pluginConfig:
        techdocs:
          builder: external
          generator:
            runIn: local
          publisher:
            awsS3:
              bucketName: '${BUCKET_NAME}'
              credentials:
```

```
          accessKeyId: '${AWS_ACCESS_KEY_ID}'
          secretAccessKey: '${AWS_SECRET_ACCESS_KEY}'
        endpoint: 'https://${BUCKET_HOST}'
        regions: '${BUCKET_REGION}'
        s3ForcePathStyle: true
      type: awsS3
```

12.1.3. Making object storage accessible to containers by using the Operator

Creating a **ObjectBucketClaim** Custom Resource (CR) automatically generates both the Developer Hub **ObjectBucketClaim** config map and secret. The config map and secret contain **ObjectBucket** access information. Adding the access information to the Operator configuration makes it accessible to the Developer Hub container by adding the following environment variables to the container:

- **BUCKET_NAME**
- **BUCKET_HOST**
- **BUCKET_PORT**
- **BUCKET_REGION**
- **BUCKET_SUBREGION**
- **AWS_ACCESS_KEY_ID**
- **AWS_SECRET_ACCESS_KEY**

These variables are then used in the TechDocs plugin configuration.

Prerequisites

- You have installed Red Hat Developer Hub on OpenShift Container Platform using the Operator.
- You have created an **ObjectBucketClaim** CR for storing files generated by TechDocs.

Procedure

- In the Developer Hub **ObjectBucketClaim** CR, enter the name of the Developer Hub **ObjectBucketClaim** config map as the value for the **spec.application.extraEnvs.configMaps** field and enter the Developer Hub **ObjectBucketClaim** secret name as the value for the **spec.application.extraEnvs.secrets** field. For example:

  ```
  spec:
    application:
      extraEnvs:
        configMaps:
          - name: <rhdh_bucket_claim_name>
        secrets:
          - name: <rhdh_bucket_claim_name>
  ```

12.1.3.1. Example TechDocs Plugin configuration for the Operator

The following example shows a Red Hat Developer Hub Operator config map configuration for the TechDocs plugin:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: dynamic-plugins-rhdh
data:
  dynamic-plugins.yaml: |
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - disabled: false
        package: ./dynamic-plugins/dist/backstage-plugin-techdocs-backend-dynamic
        pluginConfig:
          techdocs:
            builder: external
            generator:
              runIn: local
            publisher:
              awsS3:
                bucketName: '${BUCKET_NAME}'
                credentials:
                  accessKeyId: '${AWS_ACCESS_KEY_ID}'
                  secretAccessKey: '${AWS_SECRET_ACCESS_KEY}'
                endpoint: 'https://${BUCKET_HOST}'
                regions: '${BUCKET_REGION}'
                s3ForcePathStyle: true
              type: awsS3
```

12.2. CONFIGURING CI/CD TO GENERATE AND PUBLISH TECDOCS SITES

TechDocs reads the static generated documentation files from a cloud storage bucket, such as OpenShift Data Foundation. The documentation site is generated on the CI/CD workflow associated with the repository containing the documentation files. You can generate docs on CI and publish to a cloud storage using the **techdocs-cli** CLI tool.

You can use the following example to create a script for TechDocs publication:

```
# Prepare
REPOSITORY_URL='https://github.com/org/repo'
git clone $REPOSITORY_URL
cd repo

# Install @techdocs/cli, mkdocs and mkdocs plugins
npm install -g @techdocs/cli
pip install "mkdocs-techdocs-core==1.*"

# Generate
techdocs-cli generate --no-docker
```

```
# Publish
techdocs-cli publish --publisher-type awsS3 --storage-name <bucket/container> --entity
<Namespace/Kind/Name>
```

The TechDocs workflow starts the CI when a user makes changes in the repository containing the documentation files. You can configure the workflow to start only when files inside the **docs/** directory or **mkdocs.yml** are changed.

12.2.1. Preparing your repository for CI

The first step on the CI is to clone your documentation source repository in a working directory.

Procedure

- To clone your documentation source repository in a working directory, enter the following command:

  ```
  git clone <https://path/to/docs-repository/>
  ```

12.2.2. Generating the TechDocs site

Procedure

To configure CI/CD to generate your techdocs, complete the following steps:

1. Install the **npx** package to run **techdocs-cli** using the following command:

   ```
   npm install -g npx
   ```

2. Install the **techdocs-cli** tool using the following command:

   ```
   npm install -g @techdocs/cli
   ```

3. Install the **mkdocs** plugins using the following command:

   ```
   pip install "mkdocs-techdocs-core==1.*"
   ```

4. Generate your techdocs site using the following command:

   ```
   npx @techdocs/cli generate --no-docker --source-dir <path_to_repo> --output-dir ./site
   ```

   Where **<path_to_repo>** is the location in the file path that you used to clone your repository.

12.2.3. Publishing the TechDocs site

Procedure

To publish your techdocs site, complete the following steps:

1. Set the necessary authentication environment variables for your cloud storage provider.

2. Publish your techdocs using the following command:

```
npx @techdocs/cli publish --publisher-type <awsS3|googleGcs> --storage-name
<bucket/container> --entity <namespace/kind/name> --directory ./site
```

3. Add a **.github/workflows/techdocs.yml** file in your Software Template(s). For example:

```
name: Publish TechDocs Site

on:
 push:
   branches: [main]
   # You can even set it to run only when TechDocs related files are updated.
   # paths:
   #   - "docs/**"
   #   - "mkdocs.yml"

jobs:
 publish-techdocs-site:
   runs-on: ubuntu-latest

   # The following secrets are required in your CI environment for publishing files to AWS S3.
   # e.g. You can use GitHub Organization secrets to set them for all existing and new
repositories.
   env:
     TECHDOCS_S3_BUCKET_NAME: ${{ secrets.TECHDOCS_S3_BUCKET_NAME }}
     AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
     AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
     AWS_REGION: ${{ secrets.AWS_REGION }}
     ENTITY_NAMESPACE: 'default'
     ENTITY_KIND: 'Component'
     ENTITY_NAME: 'my-doc-entity'
     # In a Software template, Scaffolder will replace {{cookiecutter.component_id | jsonify}}
     # with the correct entity name. This is same as metadata.name in the entity's catalog-
info.yaml
     # ENTITY_NAME: '{{ cookiecutter.component_id | jsonify }}'

   steps:
     - name: Checkout code
       uses: actions/checkout@v3

     - uses: actions/setup-node@v3
     - uses: actions/setup-python@v4
       with:
         python-version: '3.9'

     - name: Install techdocs-cli
       run: sudo npm install -g @techdocs/cli

     - name: Install mkdocs and mkdocs plugins
       run: python -m pip install mkdocs-techdocs-core==1.*

     - name: Generate docs site
       run: techdocs-cli generate --no-docker --verbose

     - name: Publish docs site
```

```
      run: techdocs-cli publish --publisher-type awsS3 --storage-name
$TECHDOCS_S3_BUCKET_NAME --entity
$ENTITY_NAMESPACE/$ENTITY_KIND/$ENTITY_NAME
```