Red Hat Developer Hub 1.8

Orchestrator in Red Hat Developer Hub

Orchestrator enables serverless workflows for cloud migration, onboarding, and customization in Red Hat Developer Hub

Last Updated: 2026-01-27

Red Hat Developer Hub 1.8 Orchestrator in Red Hat Developer Hub

Orchestrator enables serverless workflows for cloud migration, onboarding, and customization in Red Hat Developer Hub

Legal Notice

Abstract

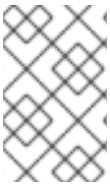
As an administrator, you can use Orchestrator to enable serverless workflows in Red Hat Developer Hub to support cloud migration, developer onboarding, and custom workflows.

Table of Contents

CHAPTER 1. ABOUT ORCHESTRATOR IN RED HAT DEVELOPER HUB

You can streamline and automate your work by using the Orchestrator in Red Hat Developer Hub. It enables you to:

- Design, run, and monitor workflows to simplify multi-step processes across applications and services.
- Standardize onboarding, migration, and integration workflows to reduce manual effort and improve consistency.
- Extend RHDH with enterprise-grade Orchestration features to support collaboration and scalability.



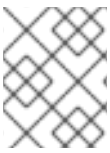
NOTE

Orchestrator currently supports only Red Hat OpenShift Container Platform (OpenShift Container Platform); it is not available on Microsoft Azure Kubernetes Service (AKS), Amazon Elastic Kubernetes Service (EKS), or Google Kubernetes Engine (GKE).

1.1. COMPATIBILITY GUIDE FOR ORCHESTRATOR

The following table lists the RHDH Orchestrator plugin versions and their compatible corresponding infrastructure versions.

Orchestrator plugin version	Red Hat Developer Hub (RHDH) version	OpenShift version	OpenShift Serverless Logic (OSL) version	OpenShift Serverless version
Orchestrator **1.5**	**1.5**	**4.14 - 4.18**	OSL **1.35**	**1.35**
Orchestrator **1.6**	**1.6**	**4.14 - 4.18**	OSL **1.36**	**1.36**
Orchestrator **1.7.1**	**1.7**	**4.16 - 4.19**	OSL **1.36**	**1.36**
Orchestrator 1.8.2	1.8	**4.16 - 4.19**	OSL **1.36**	**1.36**



NOTE

Orchestrator plugin supports the same OpenShift Container Platform versions as RHDH. See the Life Cycle page.

1.2. UNDERSTAND ORCHESTRATOR ARCHITECTURE

The Orchestrator architecture is composed of several components, each contributing to the running and management of workflows.

Red Hat Developer Hub (RHDH)

Serves as the primary interface. It contains the following subcomponents:

Orchestrator frontend plugins

Provide the interface for users to run and monitor workflows within RHDH.

Orchestrator backend plugins

Get workflow data into Developer Hub.

Notifications plugins

Inform users about workflow events.

Openshift Serverless Logic Operator

Serves as the workflow engine, and its subcomponents handle running, executing and providing persistence for the workflows. The Red Hat Developer Hub Operator and the Red Hat Developer Hub Helm chart manage the following lifecycle of these subcomponents:

Sonataflow Runtime/Workflow Application

Functions as a deployed workflow. Operates as an HTTP server, handling requests for running workflow instances. It is managed as a Kubernetes (K8s) deployment by the Openshift Serverless Logic Operator.

Data Index Service

Serves as a repository for workflow definitions, instances, and associated jobs. It exposes a GraphQL API used by the Orchestrator backend plugin to retrieve workflow definitions and instances.

Job Service

Orchestrates scheduled tasks for workflows.

OpenShift Serverless

Provides serverless capabilities essential for workflow communication. It employs Knative eventing to interface with the Data Index service and uses Knative functions to introduce more complex logic to workflows.

PostgreSQL Server

Provides a database solution essential for data persistence within the Orchestrator ecosystem. The system uses PostgreSQL Server for storing both Sonataflow information and Developer Hub data.

OpenShift AMQ Streams (Strimzi/Kafka)

Provides enhanced reliability of the eventing system. Eventing can work without Kafka by using direct HTTP calls, however, this approach is not reliable.
Optional: The current deployment iteration does not natively integrate or include the AMQ Streams Operator. However, you can add the Operator post-install for enhanced reliability if you require it.

1.3. GETTING STARTED WITH ORCHESTRATOR

To start using Orchestrator in RHDH, you must:

- Install the required infrastructure components, such as OpenShift Serverless Operator, and OpenShift Serverless Logic Operator
- Configure your Backstage custom resource (CR) or Helm values file for Orchestrator

NOTE

When using the RHDH Operator, you must first install the required infrastructure components. The Operator then provisions the dependent SonataFlow resources once the Orchestrator plugins are enabled in the Backstage CR.

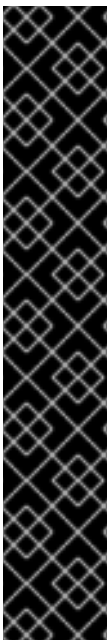
When using the RHDH Helm chart, the required infrastructure components are installed automatically using the dedicated **redhat-developer-hub-orchestrator-infra** Helm chart prior to enabling the Orchestrator plugins in the main RHDH chart.

1.4. ORCHESTRATOR PLUGIN DEPENDENCIES FOR OPERATOR INSTALLATION

When you enable the Orchestrator plugin in your Backstage custom resource (CR), the Operator automatically provisions the following required dependencies:

- A **SonataflowPlatform** CR
- **NetworkPolicies** that allow traffic between infrastructure resources (Knative, Serverless Logic Operator), monitoring traffic, and intra-namespace traffic

The Orchestrator plugin requires these components to run. For example, to communicate with the SonataFlow platform, the Orchestrator plugin uses the **sonataflow-platform-data-index-service**, which is created by the **SonataFlowPlatform** CR.



IMPORTANT

The **SonataFlowPlatform** CR contains Data Index service that requires PostgreSQL database as shown in the following example:

```
    persistence:
      postgresql:
        secretRef:
          name: backstage-psql-secret-{{backstage-name}}
          userKey: POSTGRES_USER
          passwordKey: POSTGRES_PASSWORD
        serviceRef:
          name: backstage-psql-{{backstage-name}} # # Namespace where the
Backstage CR is created
          namespace: {{backstage-ns}} # Namespace where the Backstage (CR) is
created
          databaseName: backstage_plugin_orchestrator
```

By default, the Orchestrator plugin dependencies use the following:

- The PostgreSQL database named **backstage_plugin_orchestrator** created by Backstage
- A Secret created by Backstage Operator for the PostgreSQL with **POSTGRES_USER** and **POSTGRES_PASSWORD** keys as the database credentials in the Backstage CR namespace.
- A Service created by Backstage Operator for the PostgreSQL database with the name **backstage-psql-{{backstage-name}}** in the Backstage CR namespace.

NOTE

To enable the Backstage Operator to work with the SonataFlow platform, its **ServiceAccount** must have the appropriate permissions.

The Operator automatically creates the required Role and *RoleBinding* resource in **profile/rhdh/plugin-rbac** directory.

Additional resources

- Dynamic plugins dependency management

CHAPTER 2. ENABLING ORCHESTRATOR PLUGINS COMPONENTS

2.1. ENABLING ORCHESTRATOR PLUGINS COMPONENTS

To use the Orchestrator, enable the following Orchestrator plugins for Red Hat Developer Hub, that are disabled by default:

Orchestrator-frontend plugin

backstage-plugin-orchestrator

Provides the interface for users to run and monitor workflows within RHDH. You can run and track the execution status of processes.

Orchestrator-backend plugin

backstage-plugin-orchestrator-backend

Gets workflow data into Developer Hub making sure RHDH processes critical workflow metadata and runtime status fulfilling your need for visibility.

Orchestrator-form-widget

backstage-plugin-orchestrator-form-widgets

Provides custom widgets for the workflow execution form, allowing you to customize input fields and streamline the process of launching workflows.

Orchestrator-scaffolder-backend-module

scaffolder-backend-module-orchestrator

Provides callable actions from Scaffolder templates, such as **orchestrator:workflow:run** or **orchestrator:workflow:get_params**.

Prerequisites

- You have installed the following operators:
  - Openshift Serverless
  - Openshift Serverless Logic (OSL)
- (Optional) For managing the Orchestrator project, you have an instance of Argo CD or Red Hat OpenShift GitOps in the cluster. It is disabled by default.
- (Optional) To use Tekton tasks and the build pipeline, you have an instance of Tekton or Red Hat OpenShift Pipelines in the cluster. These features are disabled by default.

Procedure

- Locate your Developer Hub configuration and enable the Orchestrator plugins and the supporting notification plugins.

```
plugins:
  - package: "@redhat/backstage-plugin-orchestrator@1.8.2"
```

```
    disabled: false
  - package: "@redhat/backstage-plugin-orchestrator-backend-dynamic@1.8.2"
    disabled: false
  - package: "@redhat/backstage-plugin-scaffolder-backend-module-orchestrator-
dynamic@1.8.2"
    disabled: false
  - package: "@redhat/backstage-plugin-orchestrator-form-widgets@1.8.2"
    disabled: false
  - package: "./dynamic-plugins/dist/backstage-plugin-notifications"
    disabled: false
  - package: "./dynamic-plugins/dist/backstage-plugin-signals"
    disabled: false
  - package: "./dynamic-plugins/dist/backstage-plugin-notifications-backend-dynamic"
    disabled: false
  - package: "./dynamic-plugins/dist/backstage-plugin-signals-backend-dynamic"
    disabled: false
```

CHAPTER 3. INSTALLING RED HAT DEVELOPER HUB WITH ORCHESTRATOR BY USING THE RED HAT DEVELOPER HUB OPERATOR

You can install Red Hat Developer Hub with Orchestrator by using the Red Hat Developer Hub Operator.

3.1. ENABLING THE ORCHESTRATOR PLUGIN USING THE OPERATOR

You can enable the Orchestrator plugin in RHDH by configuring dynamic plugins in your Backstage custom resource (CR).

Prerequisites

- You have installed RHDH on OpenShift Container Platform.
- You have access to edit or create ConfigMaps in the namespace where the Backstage CR is deployed.

Procedure

1. To enable the Orchestrator plugin with default settings, set **disabled: false** for the package. For example, **package: "@redhat/backstage-plugin-orchestrator@<plugin_version>** is set to **disabled: false**:

```
- package: "@redhat/backstage-plugin-orchestrator@<plugin_version>"
  disabled: false
```

NOTE

When you enable the plugins, the pre-loaded plugin configuration are used. Additionally, the **ref: sonataflow** field installs the Openshift Serverless and Openshift Serverless Logic resources. This happens automatically when you are using the Operator.

Example: Complete configuration of the Orchestrator plugin

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: orchestrator-plugin
data:
    dynamic-plugins.yaml: |
      includes:
        - dynamic-plugins.default.yaml
      plugins:
        - package: "@redhat/backstage-plugin-orchestrator@1.8.2"
          disabled: false
          pluginConfig:
            dynamicPlugins:
                frontend:
                  red-hat-developer-hub.backstage-plugin-orchestrator:
```

```
                appIcons:
                  - importName: OrchestratorIcon
                    name: orchestratorIcon
                dynamicRoutes:
                  - importName: OrchestratorPage
                    menuItem:
                      icon: orchestratorIcon
                      text: Orchestrator
                    path: /orchestrator
                entityTabs:
                  - path: /workflows
                    title: Workflows
                    mountPoint: entity.page.workflows
                mountPoints:
                  - mountPoint: entity.page.workflows/cards
                  importName: OrchestratorCatalogTab
                  config:
                    layout:
                      gridColumn: '1 / -1'
                      if:
                        anyOf:
                          - IsOrchestratorCatalogTabAvailable
        - package: "@redhat/backstage-plugin-orchestrator-backend-dynamic@1.8.2"
          disabled: false
          pluginConfig:
            orchestrator:
              dataIndexService:
                url: http://sonataflow-platform-data-index-service
          dependencies:
            - ref: sonataflow
        - package: "@redhat/backstage-plugin-scaffolder-backend-module-orchestrator-
dynamic@1.8.2"
          disabled: false
          pluginConfig:
            orchestrator:
              dataIndexService:
                url: http://sonataflow-platform-data-index-service
        - package: "@redhat/backstage-plugin-orchestrator-form-widgets@1.8.2"
          disabled: false
          pluginConfig:
            dynamicPlugins:
              frontend:
                red-hat-developer-hub.backstage-plugin-orchestrator-form-widgets: { }
---
apiVersion: rhdh.redhat.com/v1alpha3
kind: Backstage
metadata:
  name: orchestrator
spec:
  application:
    appConfig:
      configMaps:
        - name: app-config-rhdh
    dynamicPluginsConfigMapName: orchestrator-plugin
```

2. Create a secret containing the **BACKEND_SECRET** value as shown in the following example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config-rhdh
data:
  app-config.yaml: |-
    auth:
      environment: development
      providers:
        guest:
          # using the guest user to query the '/api/dynamic-plugins-info/loaded-plugins' endpoint.
          dangerouslyAllowOutsideDevelopment: true
    backend:
      auth:
        externalAccess:
          - type: static
            options:
              token: ${BACKEND_SECRET}
              subject: orchestrator
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-auth-secret
stringData:
  # generated with the command below (from https://backstage.io/docs/auth/service-to-service-auth/#setup):
  # node -p 'require("crypto").randomBytes(24).toString("base64")'
  # notsecret
  BACKEND_SECRET: "R2FxRVNrcmwzYzhhN3l0V1VRcnQ3L1pLT09WaVhDNUEK"
```

3. Configure your Backstage CR to update the secret name in the **extraEnvs** field as shown in the following example:

```
apiVersion: rhdh.redhat.com/v1alpha4
kind: Backstage
metadata:
  name: orchestrator
spec:
  application:
    appConfig:
      configMaps:
        - name: app-config-rhdh
    dynamicPluginsConfigMapName: orchestrator-plugin
    extraEnvs:
      secrets:
        # secret that contains the BACKEND_SECRET key
        - name: backend-auth-secret
```

Verification

- In the RHDH console, confirm that the Orchestrator frontend and backend features are available.

3.2. UPGRADING THE ORCHESTRATOR PLUGIN FROM 1.7 TO 1.8

You can upgrade an existing **1.7** Operator-backed instance with Orchestrator enabled by upgrading the Red Hat Developer Hub Operator to 1.8. After upgrading the Operator to 1.8, manually update the **dynamic-plugins** ConfigMap to set the Orchestrator plugin versions to **1.8.2**.

Prerequisites

- You have have a running instance of Red Hat Developer Hub with Orchestrator **1.7** backed by the Operator.
- You have upgraded the Red Hat Developer Hub Operator to 1.8.

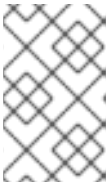
Procedure

1. Update your **dynamic-plugins** ConfigMap to set the version of the Orchestrator plugin to **1.8.2**. The following YAML configuration is an example of a **dynamic-plugins** ConfigMap enabling the Orchestrator plugins in RHDH for Operator-backed instances:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: dynamic-plugins-rhdh
data:
  dynamic-plugins.yaml: |
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: "@redhat/backstage-plugin-orchestrator@1.8.2"
        disabled: false
      - package: "@redhat/backstage-plugin-orchestrator-backend-dynamic@1.8.2"
        disabled: false
        dependencies:
          - ref: sonataflow
      - package: "@redhat/backstage-plugin-scaffolder-backend-module-orchestrator-
dynamic@1.8.2"
        disabled: false
      - package: "@redhat/backstage-plugin-orchestrator-form-widgets@1.8.2"
        disabled: false
```

Verification

1. Navigate to your Red Hat Developer Hub instance.



NOTE

The upgrade takes a few minutes to complete. The Red Hat Developer Hub version does not update in the UI until all running Backstage pods are successfully recreated.

2. From the profile dropdown in the top menu, click the **Settings** icon and then locate the RHDH metadata card.

3. Confirm that the value displayed for RHDH version is 1.8.

4. Alternatively, run the following command in your terminal to wait for all pods in the current project to be in the **running** state:

```
$ oc get pods -w
```

The upgrade is successful when all Backstage-related pods show a stable **Running** status.

3.3. ORCHESTRATOR PLUGIN PERMISSIONS

The Orchestrator plugin uses the Red Hat Developer Hub permission mechanism and the Role-Based Access Control (RBAC) plugin to restrict access to backend endpoints. Orchestrator supports decoupling visibility (read) from running (update) using specific workflow IDs instead of generic permissions.

Permission name	Resource Type	Policy	Description
orchestrator.workflow	named resource	read	Lists and reads all workflow definitions. Lists and reads their instances
orchestrator.workflow.[workflowId]	named resource	read	Lists and reads a specific workflow definition. Lists and reads instances created for this particular workflow.
orchestrator.workflow.use	named resource	update	Runs or aborts any workflow.
orchestrator.workflow.use.[workflowId]	named resource	update	Runs or aborts a specific workflow.
orchestrator.workflowAdminView	named resource	read	Views instance variables and the workflow definition editor.
orchestrator.instanceAdminView	named resource	read	Views all workflow instances, including those created by other users.

WARNING

Generic permissions override specific denial policies within the same action type. To maintain granular control, avoid granting generic permissions if you intend to restrict specific workflows.

- Granting **orchestrator.workflow** (read) prevents you from denying access to **orchestrator.workflow.[workflowId]** (read).
- Granting **orchestrator.workflow.use** (update) prevents you from denying access to **orchestrator.workflow.use.[workflowId]** (update).

The [**workflowId**] must match the unique identifier in your workflow definition file. For example, in the workflow definition below, the identifier is **greeting**:

```
id: greeting
version: '1.0'
specVersion: '0.8'
name: Greeting workflow
description: YAML based greeting workflow
annotations:
  - 'workflow-type/infrastructure'
dataInputSchema: 'schemas/greeting.sw.input-schema.json'
extensions:
  - extensionid: workflow-output-schema
    outputSchema: schemas/workflow-output-schema.json
```

3.4. MANAGING ORCHESTRATOR PLUGIN PERMISSIONS USING RBAC POLICIES

You can configure Role-Based Access Control (RBAC) policies so that users can view workflow details without the permission to run those workflows. This configuration restricts user interaction to authorized workflows.

Prerequisites

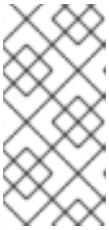
1. You have identified the [**workflowId**] for each workflow you want to restrict.
2. You have enabled the RBAC plugin.
3. You have configured the **policies-csv-file** path in your **app-config.yaml**.

Procedure

1. Identify the **workflowId** from your workflow definition file:

```
id: greeting
version: '1.0'
```

2. In your RBAC policy CSV file, define the permissions using the **p, role, permission, action, allow** format.



NOTE

Generic permissions (for example, **orchestrator.workflow**) take precedence over specific permissions targeting a **workflowId**, (for example, **orchestrator.workflow.greeting**). You cannot grant generic access and then selectively deny a specific ID.

3. Add the following example policies to your CSV file to establish basic user and administrator roles:

```
# Minimal user role - can only view and run specific workflows
p, role:default/workflowUser, orchestrator.workflow.greeting, read, allow
p, role:default/workflowUser, orchestrator.workflow.use.greeting, update, allow

# Support role - can view all workflows and instances, but not execute
p, role:default/workflowSupport, orchestrator.workflow, read, allow
p, role:default/workflowSupport, orchestrator.instanceAdminView, read, allow

# Full admin role - complete access to all Orchestrator functions
p, role:default/workflowAdmin, orchestrator.workflow, read, allow
p, role:default/workflowAdmin, orchestrator.workflow.use, update, allow
p, role:default/workflowAdmin, orchestrator.workflowAdminView, read, allow
p, role:default/workflowAdmin, orchestrator.instanceAdminView, read, allow

# Assign users to the roles
g, user:default/example_user, role:default/workflowUser
```

4. In your RHDH **app-config.yaml** file, enable permissions by adding the **orchestrator** plugin to the **rbac** section and setting **policyFileReload** to **true**.

```
permission:
  enabled: true
  rbac:
    policies-csv-file: <absolute_path_to_the_policy_file>
    pluginsWithPermission:
      - orchestrator
    policyFileReload: true
    admin:
      users:
        - name: user:default/YOUR_USER
```

5. Restart the application to apply the changes.

Verification

1. Log in as a user assigned to the **workflowUser** role.

2. Navigate to the Orchestrator plugin and verify that the workflow appears in the list.

IMPORTANT

You can view dynamic permissions containing a **workflowId** in the RBAC UI, but you cannot modify them in the interface. You must use the policy CSV file or the RBAC API to manage these specific workflow permissions.

Additional resources

- RBAC documentation.

CHAPTER 4. INSTALLING RED HAT DEVELOPER HUB WITH ORCHESTRATOR BY USING THE RED HAT DEVELOPER HUB HELM CHART

You can install Red Hat Developer Hub with Orchestrator by using the Red Hat Developer Hub Helm chart.

4.1. INSTALLING RED HAT DEVELOPER HUB (RHDH) ON OPENSHIFT CONTAINER PLATFORM WITH THE ORCHESTRATOR USING THE HELM CLI

You can install Red Hat Developer Hub (RHDH) on OpenShift Container Platform with the Orchestrator by using the Helm CLI. The installation automatically enables the required dynamic plugins and integrates workflow infrastructure.

Prerequisites

- You are logged in as an administrator and have access to the Red Hat Developer Hub Helm chart repository.
- You can install the necessary infrastructures resources, such as other Openshift operators (OpenShift Serverless and OpenShift Serverless Logic), alongside RHDH in the same namespace.
  This is a one-off requirement and must be completed before enabling the Orchestrator plugin.

Procedure

1. Manually approve the install plans for the Operators. You must run the **oc patch installplan** commands provided in the output to approve their installation.

IMPORTANT

By default, Orchestrator Infrastructure for Red Hat Developer Hub Helm chart does **not** auto-approve the required Serverless Operators. You must manually approve the install plans.

1. As an administrator, install relevant cluster-wide resources.

```
$ helm repo add openshift-helm-charts https://charts.openshift.io/
$ helm install <release_name> openshift-helm-charts/redhat-developer-hub-orchestrator-infra
```

IMPORTANT

You must be an administrator to install the **redhat-developer-hub-orchestrator-infra** Helm chart because it deploys additional cluster-scoped OpenShift Serverless and OpenShift Serverless Logic Operators. As an administrator, you must manually approve the install plans for OpenShift Serverless and Serverless Logic Operators.

2. Install the Backstage chart with the orchestrator enabled as shown in the following example:

```
$ helm install <release_name> openshift-helm-charts/redhat-developer-hub --version 1.8.2 \
  --set orchestrator.enabled=true
```

3. (Optional) Enable **Notifications** and **Signals** plugins by adding them to the **global.dynamic.plugins** list in your **values.yaml** file as shown in the following example:

```
global:
  dynamic:
    plugins:
      - disabled: false
        package: "./dynamic-plugins/dist/backstage-plugin-notifications"
      - disabled: false
        package: "./dynamic-plugins/dist/backstage-plugin-signals"
      - disabled: false
        package: "./dynamic-plugins/dist/backstage-plugin-notifications-backend-dynamic"
      - disabled: false
        package: "./dynamic-plugins/dist/backstage-plugin-signals-backend-dynamic"
```

4. (Optional) You can disable the Serverless Logic and Serverless Operators individually or together by setting their values to **false**, as shown in the following example:

```
$ helm install <release_name> openshift-helm-charts/redhat-developer-hub \
  --version 1.8.2 \
  --set orchestrator.enabled=true \
  --set orchestrator.serverlessOperator=false \
  --set orchestrator.serverlessLogicOperator=false
```

5. (Optional) If you are using an external database, add the following configuration under **orchestrator.sonataflowPlatform** in your **values.yaml** file:

```
orchestrator:
  sonataflowPlatform:
    externalDBsecretRef: "<cred-secret>"
    externalDBName: "<database_name>" # The name of the user-configured existing
database (Not the database that the orchestrator and sonataflow resources use).
    externalDBHost: "<database_host>"
    externalDBPort: "<database_port>"
```

NOTE

This step only configures the Orchestrators use of an external database. To configure Red Hat Developer Hub to use an external PostgreSQL instance, follow the steps in Configuring a PostgreSQL instance using Helm .

Verification

1. Verify that the Orchestrator plugin is visible in the Red Hat Developer Hub UI.
2. Create and run sample workflows to confirm the orchestration is functioning correctly.

4.2. INSTALL RED HAT DEVELOPER HUB (RHDH) USING HELM FROM THE OPENSHIFT CONTAINER PLATFORM WEB CONSOLE

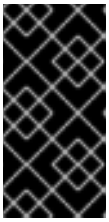
You can install Red Hat Developer Hub (RHDH) with the Orchestrator by using the (OpenShift Container Platform) web console. This method is useful if you prefer a graphical interface or want to deploy cluster-wide resources without using the Helm CLI.

Prerequisites

- You are logged in to the OpenShift Container Platform web console as an administrator.
- You have access to the Red Hat Developer Hub Helm chart repository.
- Your cluster has internet access or the Helm charts are mirrored in a disconnected environment.

Procedure

1. In the OpenShift Container Platform web console, go to the Helm Charts and verify that the Red Hat Developer Hub Helm chart repository is available.
2. Search for the Orchestrator infrastructure for Red Hat Developer Hub and select **Install**.



IMPORTANT

You must be an administrator to install the Orchestrator Infrastructure for Red Hat Developer Hub Helm chart because it deploys cluster-scoped resources. As an administrator, you must manually approve the install plans for OpenShift Serverless and Serverless Logic Operators.

As a regular user, search for the Red Hat Developer Hub chart and install it by setting the value of **orchestrator.enabled** to **true**. Otherwise, the Orchestrator will not be deployed.

3. Wait until they are successfully deployed.
4. Monitor the deployment status by navigating to **Pods** or releases.

Verification

After deployment completes:

- The orchestrator-related pods are running in the selected namespace.
- Cluster-wide resources are present.
- You can start connecting the orchestrator to your Red Hat Developer Hub UI.

4.3. RESOURCE LIMITS FOR INSTALLING RED HAT DEVELOPER HUB WITH THE ORCHESTRATOR PLUGIN WHEN USING HELM

When installing Red Hat Developer Hub (RHDH) with the Orchestrator plugin using Helm, the chart defines default CPU and memory limits for the **SonataFlowPlatform** component.

These limits are enforced by the cluster so that pods do not exceed their allocated resources.

1. Default resource limits

Resource	Default value
CPU limits	**500m**
Memory limits	**1Gi**

1. You can override these values in any of the following ways:
   - With **values.yaml**
   - With **--set** flags
2. Override defaults with **values.yaml** as shown in the following example:

```
orchestrator:
  enabled: true
  sonataflowPlatform:
  resources:
      limits:
        cpu: "500m"
        memory: "1Gi"
```

3. Override with **--set** as shown in the following example:

```
$ helm upgrade --install <release_name>  openshift-helm-charts/redhat-developer-hub \
  --set orchestrator.enabled=true \
  --set orchestrator.sonataflowPlatform.resources.requests.cpu=500m \
  --set orchestrator.sonataflowPlatform.resources.requests.memory=128Mi \
  --set orchestrator.sonataflowPlatform.resources.limits.cpu=1 \
  --set orchestrator.sonataflowPlatform.resources.limits.memory=2Gi
```

NOTE

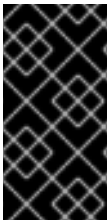
The **--set** setting is applicable only when **orchestrator.enabled** is **true**. By default, it is set to **false**.

4.4. INSTALLING ORCHESTRATOR COMPONENTS MANUALLY ON OPENSHIFT CONTAINER PLATFORM

Use manual installation when you want full control of the setup process and component versions. Manual installation method focuses on setting up the underlying infrastructure.

Procedure

1. Install the OpenShift Serverless components manually by following the instructions in the Red Hat OpenShift Serverless documentation.
2. (Optional) If required, deploy a custom PostgreSQL database.



IMPORTANT

Prevent workflow context from being lost when the Pod restarts by configuring workflow persistence. You can configure persistence at the namespace level by using the **SonataFlowPlatform** or **SonataFlow** custom resources (CR). For more information, check the Managing workflow persistence documentation.

CHAPTER 5. INSTALLING ORCHESTRATOR PLUGIN IN AN AIR-GAPPED ENVIRONMENT WITH THE OPERATOR

You can configure Red Hat Developer Hub (RHDH) with the Orchestrator plugin in a fully disconnected or partially disconnected environment by using the Operator.

5.1. INSTALLING RED HAT DEVELOPER HUB WITH ORCHESTRATOR IN A FULLY DISCONNECTED OPENSHIFT CONTAINER PLATFORM ENVIRONMENT USING THE OPERATOR

You can install Red Hat Developer Hub with Orchestrator plugin in a fully air-gapped environment using the Operator.

A disconnected installation prevents unauthorized access, data transfer, or communication with external sources.

You can use the helper script to install Red Hat Developer Hub by mirroring the Operator-related images to disk and transferring them to your disconnected environment without any connection to the internet.

Prerequisites

- You have mirrored the Red Hat Developer Hub Operator images to the local registry using the RHDH mirroring script. For more information, see *Installing Red Hat Developer Hub in a fully disconnected environment with the Operator*.
- You have set up your disconnected environment using a local registry.
- You have permissions to push NPM packages to an NPM server available in your restricted network.
- You have installed the **oc-mirror** tool, with a version corresponding to the version of your OpenShift Container Platform cluster.

Procedure

1. Create an **ImageSetConfiguration** file for **oc-mirror**. You must include the images and operators required by the Serverless Logic Operator in the **ImageSetConfiguration** file, as **oc-mirror** does not automatically mirror all images. Use the following example:

```
apiVersion: mirror.openshift.io/v2alpha1
kind: ImageSetConfiguration
mirror:
  additionalimages:
  - name: registry.redhat.io/openshift-serverless-1/logic-jobs-service-postgresql-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-jobs-service-ephemeral-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-data-index-postgresql-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-data-index-ephemeral-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-db-migrator-tool-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-swf-builder-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-swf-devmode-rhel8:1.36.0
  operators:
    - catalog: registry.redhat.io/redhat/redhat-operator-index:4.19
      # For example: registry.redhat.io/redhat/redhat-operator-index:v4.19
```

```
    packages:
    - name: logic-operator-rhel8
      channels:
      - name: alpha
        minVersion: 1.36.0
        maxVersion: 1.36.0
    - name: serverless-operator
      channels:
      - name: stable
        minVersion: 1.36.0
        maxVersion: 1.36.1
```

Alternatively, you can use **podman** commands to find the missing images and add them to the **additionalimages** list if the versions change:

```
IMG=registry.redhat.io/openshift-serverless-1/logic-operator-bundle:1.36
mkdir local-manifests-osl
podman create --name temp-container "$IMG" -c "cat /manifests/logic-operator-
rhel8.clusterserviceversion.yaml"
podman cp temp-container:/manifests ./local-manifests-osl
podman rm temp-container
yq -r '.data."controllers_cfg.yaml" | from_yaml | .. | select(tag == "!!str") |
select(test("^.\\/.:.*$"))' ./local-manifests-osl/manifests/logic-operator-rhel8-controllers-
config_v1_configmap.yaml
```

2. Mirror the images in the **ImageSetConfiguration.yaml** file by running the **oc-mirror** command. For example:

```
$ oc-mirror --config=ImageSetConfiguration.yaml file:///path/to/mirror-archive --authfile
/path/to/authfile  --v2
```

NOTE

The **oc-mirror** command generates a local workspace containing the mirror archive files and the required cluster manifests.

3. Transfer the directory specified by **/path/to/mirror-archive** to a bastion host within your disconnected environment.

4. From the bastion host which has access to the mirror registry, mirror the images from the disk directory to your target registry. For example:

```
$ oc-mirror --v2 --from <mirror-archive-file> docker://<target-registry-url:port> --workspace
file://<workspace folder> --authfile /path/to/authfile
```

where:

<mirror-archive-file>

Enter the name of the transferred **tar** file.

<target-registry-url:port>

Enter your local registry, for example, **registry.localhost:5000**.

5. Apply the cluster-wide resources generated during the push step to redirect all image pulls to your local registry, as shown in the following example:

```
$ cd <workspace folder>/working-dir/cluster-resources/
$ oc apply -f .
```

6. Download the Node Package Manager (NPM) packages for orchestrator 1.8.2 using any of the following methods:

   - Download them as **tgz** files from the following registry:
     - https://npm.registry.redhat.com/@redhat/backstage-plugin-orchestrator/-/backstage-plugin-orchestrator-1.8.2.tgz
     - https://npm.registry.redhat.com/@redhat/backstage-plugin-orchestrator-backend-dynamic/-/backstage-plugin-orchestrator-backend-dynamic-1.8.2.tgz
     - https://npm.registry.redhat.com/@redhat/backstage-plugin-scaffolder-backend-module-orchestrator-dynamic/-/backstage-plugin-scaffolder-backend-module-orchestrator-dynamic-1.8.2.tgz
     - https://npm.registry.redhat.com/@redhat/backstage-plugin-orchestrator-form-widgets/-/backstage-plugin-orchestrator-form-widgets-1.8.2.tgz
   - Alternatively, use the NPM packages from Red Hat NPM registry as shown in the following example:

```
$ npm pack "@redhat/backstage-plugin-orchestrator@1.8.2" --
registry=https://npm.registry.redhat.com
$ npm pack "@redhat/backstage-plugin-orchestrator-backend-dynamic@1.8.2" --
registry=https://npm.registry.redhat.com
$ npm pack "@redhat/backstage-plugin-scaffolder-backend-module-orchestrator-
dynamic@1.8.2 --registry=https://npm.registry.redhat.com
$ npm pack "@redhat/backstage-plugin-orchestrator-form-widgets@1.8.2" --
registry=https://npm.registry.redhat.com
```

7. Push the NPM packages you have downloaded to your NPM server, as shown in the following example:

```
$ npm publish backstage-plugin-orchestrator-1.8.2.tgz
$ npm publish backstage-plugin-orchestrator-backend-dynamic-1.8.2.tgz
$ npm publish backstage-plugin-orchestrator-form-widgets-1.8.2.tgz
$ npm publish backstage-plugin-scaffolder-backend-module-orchestrator-dynamic-1.8.2.tgz
```

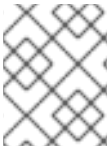
8. Install the OpenShift Serverless Operator and OpenShift Serverless Logic Operators using **OperatorHub**.

9. Create a Backstage custom resource (CR).

10. Configure the Backstage CR for the Orchestrator as described in the Orchestrator plugin dependencies for Operator installation.
    Create all the resources and configure the Backstage instance accordingly. See Configuring a custom NPM registry for instructions on how to point RHDH towards the custom NPM registry.

Verification

- Restart the RHDH pod and wait for the components to deploy properly.
- Once stable, go to the RHDH UI, and confirm that the Orchestrator UI is accessible and functioning correctly.



NOTE

The successful accessibility of the Orchestrator UI confirms that the underlying components are running and the cluster recognizes the plugin.

5.2. INSTALLING RED HAT DEVELOPER HUB WITH ORCHESTRATOR IN A PARTIALLY DISCONNECTED OPENSHIFT CONTAINER PLATFORM ENVIRONMENT USING THE OPERATOR

You can install Red Hat Developer Hub with Orchestrator plugin in a partial air-gapped environment using the Operator.

A disconnected installation prevents unauthorized access, data transfer, or communication with external sources.

You can use the **oc-mirror** command to mirror resources directly to your accessible local mirror registry and apply the generated cluster resources.

Prerequisites

- You have mirrored the Red Hat Developer Hub Operator images to the local registry using the RHDH mirroring script. For more information, see *Installing Red Hat Developer Hub in a partially disconnected environment with the Operator*.
- You have set up your disconnected environment using a local registry.
- You have permissions to push NPM packages to an NPM server available in your restricted network.
- You have installed the **oc-mirror** tool, with a version corresponding to the version of your OpenShift Container Platform cluster.

Procedure

1. Create an **ImageSetConfiguration** file for **oc-mirror**. You must include the images and operators required by the Serverless Logic Operator in the **ImageSetConfiguration** file, as **oc-mirror** does not automatically mirror all images. Use the following example:

```
apiVersion: mirror.openshift.io/v2alpha1
kind: ImageSetConfiguration
mirror:
  additionalimages:
  - name: registry.redhat.io/openshift-serverless-1/logic-jobs-service-postgresql-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-jobs-service-ephemeral-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-data-index-postgresql-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-data-index-ephemeral-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-db-migrator-tool-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-swf-builder-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-swf-devmode-rhel8:1.36.0
```

```
operators:
  - catalog: registry.redhat.io/redhat/redhat-operator-index:4.19
    # For example: registry.redhat.io/redhat/redhat-operator-index:v4.19
    packages:
    - name: logic-operator-rhel8
      channels:
      - name: alpha
        minVersion: 1.36.0
        maxVersion: 1.36.0
    - name: serverless-operator
      channels:
      - name: stable
        minVersion: 1.36.0
        maxVersion: 1.36.1
```

Alternatively, you can use the **podman** commands to find the missing images and add them to the **additionalimages** list if the versions change:

```
IMG=registry.redhat.io/openshift-serverless-1/logic-operator-bundle:1.36.0-8
mkdir local-manifests-osl
podman create --name temp-container "$IMG" -c "cat /manifests/logic-operator-
rhel8.clusterserviceversion.yaml"
podman cp temp-container:/manifests ./local-manifests-osl
podman rm temp-container
yq -r '.data."controllers_cfg.yaml" | from_yaml | .. | select(tag == "!!str") |
select(test("^.\\/.:.*$"))' ./local-manifests-osl/manifests/logic-operator-rhel8-controllers-
config_v1_configmap.yaml
```

2. Mirror the images in the **ImageSetConfiguration.yaml** file by running the **oc-mirror** command. For example:

```
$ oc-mirror --config=imagesetconfiguration.yaml docker://<registry URL:port> --workspace
file://<workspace folder> --authfile /path/to/authfile  --v2
$ cd <workspace folder>/working-dir/cluster-resources/
$ oc apply -f .
```

3. Download the Node Package Manager (NPM) packages for orchestrator 1.8.2 using any of the following methods:

   - Download them as **tgz** files from the following registry:
     - https://npm.registry.redhat.com/@redhat/backstage-plugin-orchestrator/-/backstage-plugin-orchestrator-1.8.2.tgz
     - https://npm.registry.redhat.com/@redhat/backstage-plugin-orchestrator-backend-dynamic/-/backstage-plugin-orchestrator-backend-dynamic-1.8.2.tgz
     - https://npm.registry.redhat.com/@redhat/backstage-plugin-scaffolder-backend-module-orchestrator-dynamic/-/backstage-plugin-scaffolder-backend-module-orchestrator-dynamic-1.8.2.tgz
     - https://npm.registry.redhat.com/@redhat/backstage-plugin-orchestrator-form-widgets/-/backstage-plugin-orchestrator-form-widgets-1.8.2.tgz
   - Alternatively, use the NPM packages from Red Hat NPM registry as shown in the following example:

```
$ npm pack "@redhat/backstage-plugin-orchestrator@1.8.2" --
registry=https://npm.registry.redhat.com
$ npm pack "@redhat/backstage-plugin-orchestrator-backend-dynamic@1.8.2" --
registry=https://npm.registry.redhat.com
$ npm pack "@redhat/backstage-plugin-scaffolder-backend-module-orchestrator-
dynamic@1.8.2 --registry=https://npm.registry.redhat.com
$ npm pack "@redhat/backstage-plugin-orchestrator-form-widgets@1.8.2" --
registry=https://npm.registry.redhat.com
```

4. Push the NPM packages you have downloaded to your NPM server, as shown in the following example:

```
$ npm publish backstage-plugin-orchestrator-1.8.2.tgz
$ npm publish backstage-plugin-orchestrator-backend-dynamic-1.8.2.tgz
$ npm publish backstage-plugin-orchestrator-form-widgets-1.8.2.tgz
$ npm publish backstage-plugin-scaffolder-backend-module-orchestrator-dynamic-1.8.2.tgz
```

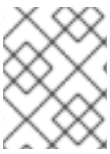
5. Install the OpenShift Serverless Operator and OpenShift Serverless Logic Operators using **OperatorHub**.

6. Create a Backstage custom resource (CR).

7. Configure the Backstage CR for the Orchestrator as described in the Orchestrator plugin dependencies for Operator installation.
   Create all the resources and configure the Backstage instance accordingly. See Configuring a custom NPM registry for instructions on how to point RHDH towards the custom NPM registry.

Verification

- Restart the RHDH pod and wait for the components to deploy properly.
- Once stable, go to the RHDH UI, and confirm that the Orchestrator UI is accessible and functioning correctly.



NOTE

The successful accessibility of the Orchestrator UI confirms that the underlying components are running and the cluster recognizes the plugin.

CHAPTER 6. INSTALLING ORCHESTRATOR PLUGIN IN AN AIR-GAPPED ENVIRONMENT WITH THE HELM CHART

You can configure Red Hat Developer Hub (RHDH) with the Orchestrator plugin in a fully disconnected or partially disconnected environment by using the Helm chart.

6.1. INSTALLING RED HAT DEVELOPER HUB WITH ORCHESTRATOR IN A FULLY DISCONNECTED OPENSHIFT CONTAINER PLATFORM ENVIRONMENT USING THE HELM CHART

You can install Red Hat Developer Hub (RHDH) with the Orchestrator plugin in a fully air-gapped OpenShift Container Platform environment using the Helm chart.

You can mirror images to an intermediary disk, and then mirror from the disk to your target local registry and apply the generated cluster resources.

Prerequisites

- You have set up your disconnected environment using a local registry.
- You have permissions to push NPM packages to an NPM server available in your restricted network.
- You have installed the **oc-mirror** tool, with a version corresponding to the version of your OpenShift Container Platform cluster.

Procedure

1. Create an **ImageSetConfiguration.yaml** file for **oc-mirror**. You must use an **ImageSetConfiguration** file to include all mirrored images required by the Serverless Logic Operator, as shown in the following example:

```
apiVersion: mirror.openshift.io/v2alpha1
kind: ImageSetConfiguration
mirror:
  additionalimages:
  - name: registry.redhat.io/openshift-serverless-1/logic-jobs-service-postgresql-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-jobs-service-ephemeral-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-data-index-postgresql-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-data-index-ephemeral-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-db-migrator-tool-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-swf-builder-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-swf-devmode-rhel8:1.36.0

  helm:
    repositories:
      - name: openshift-charts
        url: https://charts.openshift.io
        charts:
          - name: redhat-developer-hub
            version: "1.8.2"
          - name: redhat-developer-hub-orchestrator-infra
            version: "1.8.2"
```

```
operators:
  - catalog: registry.redhat.io/redhat/redhat-operator-index:4.19
    # For example: registry.redhat.io/redhat/redhat-operator-index:v4.19
    packages:
    - name: logic-operator-rhel8
      channels:
      - name: alpha
        minVersion: 1.36.0
        maxVersion: 1.36.0
    - name: serverless-operator
      channels:
      - name: stable
        minVersion: 1.36.0
        maxVersion: 1.36.1
```

Alternatively, you can use **podman** commands to find the missing images and add them to the **additionalimages** list if the versions change:

```
IMG=registry.redhat.io/openshift-serverless-1/logic-operator-bundle:1.36
mkdir local-manifests-osl
podman create --name temp-container "$IMG" -c "cat /manifests/logic-operator-
rhel8.clusterserviceversion.yaml"
podman cp temp-container:/manifests ./local-manifests-osl
podman rm temp-container
yq -r '.data."controllers_cfg.yaml" | from_yaml | .. | select(tag == "!!str") |
select(test("^.\\/.:.*$"))' ./local-manifests-osl/manifests/logic-operator-rhel8-controllers-
config_v1_configmap.yaml
```

2. Mirror the images in the **ImageSetConfiguration.yaml** file by running the **oc-mirror** command. For example:

```
$ oc-mirror --config=ImageSetConfiguration.yaml file:///path/to/mirror-archive --authfile
/path/to/authfile  --v2
```

NOTE

The **oc-mirror** command pulls the charts listed in the **ImageSetConfiguration** file and makes them available as **tgz** archives under the **/path/to/mirror-archive** directory.

3. Apply the cluster-wide resources generated during the push step to redirect all image pulls to your local registry, as shown in the following example:

```
$ cd <workspace folder>/working-dir/cluster-resources/
$ oc apply -f .
```

4. Transfer the generated mirror archive file, for example, **/path/to/mirror-archive/mirror_000001.tar**, to a bastion host within your disconnected environment.

5. From the bastion host in your disconnected environment, which has access to the mirror registry, mirror the images from the archive file to your target registry. For example:

```
$ oc-mirror --v2 --from <mirror-archive-file> docker://<target-registry-url:port> --workspace
file://<workspace folder> --authfile /path/to/authfile
```

where:

<mirror-archive-file>

Enter the name of the transferred **tar** file.

<target-registry-url:port>

Enter your local registry, for example, **registry.localhost:5000**.

6. Download the Node Package Manager (NPM) packages for orchestrator 1.8.2 using any of the following methods:

   - Download them as **tgz** files from the following registry:
     - https://npm.registry.redhat.com/@redhat/backstage-plugin-orchestrator/-/backstage-plugin-orchestrator-1.8.2.tgz
     - https://npm.registry.redhat.com/@redhat/backstage-plugin-orchestrator-backend-dynamic/-/backstage-plugin-orchestrator-backend-dynamic-1.8.2.tgz
     - https://npm.registry.redhat.com/@redhat/backstage-plugin-scaffolder-backend-module-orchestrator-dynamic/-/backstage-plugin-scaffolder-backend-module-orchestrator-dynamic-1.8.2.tgz
     - https://npm.registry.redhat.com/@redhat/backstage-plugin-orchestrator-form-widgets/-/backstage-plugin-orchestrator-form-widgets-1.8.2.tgz
   - Alternatively, use the NPM packages from Red Hat NPM registry as shown in the following example:

     ```
     $ npm pack "@redhat/backstage-plugin-orchestrator@1.8.2" --
     registry=https://npm.registry.redhat.com
     $ npm pack "@redhat/backstage-plugin-orchestrator-backend-dynamic@1.8.2" --
     registry=https://npm.registry.redhat.com
     $ npm pack "@redhat/backstage-plugin-scaffolder-backend-module-orchestrator-
     dynamic@1.8.2 --registry=https://npm.registry.redhat.com
     $ npm pack "@redhat/backstage-plugin-orchestrator-form-widgets@1.8.2" --
     registry=https://npm.registry.redhat.com
     ```

7. Push the NPM packages you have downloaded to your NPM server, as shown in the following example:

   ```
   $ npm publish backstage-plugin-orchestrator-1.8.2.tgz
   $ npm publish backstage-plugin-orchestrator-backend-dynamic-1.8.2.tgz
   $ npm publish backstage-plugin-orchestrator-form-widgets-1.8.2.tgz
   $ npm publish backstage-plugin-scaffolder-backend-module-orchestrator-dynamic-1.8.2.tgz
   ```

8. Apply the **redhat-developer-hub-orchestrator-infra** Helm chart and approve the install plans. See Air-gapped installation with Helm chart instructions for details.

9. Apply the RHDH 1.8 Helm chart. Include the version 1.8.2 and enable the Orchestrator plugin, as shown in the following example:

```
orchestrator.enabled=true
```

10. The RHDH 1.8 Helm chart defaults to pulling Orchestrator plugins from the official Red Hat NPM registry using full URL references. You must override this behavior to point to your local registry. To configure the Orchestrator plugins to use a custom registry, complete the following steps:

    - Open your **values.yaml** file.
    - List the Orchestrator plugin packages under the **orchestrator.plugins** section. You must replace the simplified package references with the full URLs that point to your custom NPM registry, as shown in the following example:

```
orchestrator:
  plugins:
    - disabled: false
      package: <custom_NPM_registry_URL>[:<port>]/@redhat/backstage-plugin-orchestrator-backend-dynamic/-/backstage-plugin-orchestrator-backend-dynamic-{product-bundle-version}.tgz
      integrity: sha512-xxxxxx
    - disabled: false
      package: <custom_NPM_registry_URL>[:<port>]/@redhat/backstage-plugin-orchestrator/-/backstage-plugin-orchestrator-{product-bundle-version}.tgz
      integrity: sha512-xxxxxy
    - disabled: false
      package: <custom_NPM_registry_URL>[:<port>]/@redhat/backstage-plugin-orchestrator-form-widgets/-/backstage-plugin-orchestrator-form-widgets-{product-bundle-version}.tgz
      integrity: sha512-xxxxxz
    - disabled: false
      package: <custom_NPM_registry_URL>[:<port>]/@redhat/backstage-plugin-scaffolder-backend-module-orchestrator-dynamic/-/backstage-plugin-scaffolder-backend-module-orchestrator-dynamic-{product-bundle-version}.tgz
      integrity: sha512-xxxx1
```

    where:

    <custom_NPM_registry_URL>

    Enter the address of your custom registry and make sure the integrity checksum, such as sha512-xxxxxx, matches the files in your registry.

Verification

- Restart the RHDH Pod and wait for the components to deploy properly.
- After deployment is complete, go to the **RHDH** UI and confirm that the **Orchestrator** UI is accessible and functioning correctly.

NOTE

The successful accessibility of the Orchestrator UI confirms that the underlying components are running and the cluster recognizes the plugin.

6.2. INSTALLING RED HAT DEVELOPER HUB WITH ORCHESTRATOR IN A PARTIALLY DISCONNECTED OPENSHIFT CONTAINER PLATFORM ENVIRONMENT USING THE HELM CHART

You can install Red Hat Developer Hub (RHDH) with the Orchestrator plugin in a partial OpenShift Container Platform environment using the Helm chart.

A disconnected installation prevents unauthorized access, data transfer, or communication with external sources.

You can use the **oc-mirror** command to mirror resources directly to your accessible local registry and apply the generated cluster resources.

Prerequisites

- You have set up your disconnected environment using a local registry.
- You have permissions to push NPM packages to an NPM server available in your restricted network.
- You have installed the **oc-mirror** tool, with a version corresponding to the version of your OpenShift Container Platform cluster.

Procedure

1. Create an **ImageSetConfiguration** file for **oc-mirror**. You must include the images and operators required by the Serverless Logic Operator in the **ImageSetConfiguration** file, as **oc-mirror** does not automatically mirror all images. Use the following example:

```
apiVersion: mirror.openshift.io/v2alpha1
kind: ImageSetConfiguration
mirror:
  additionalimages:
  - name: registry.redhat.io/openshift-serverless-1/logic-jobs-service-postgresql-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-jobs-service-ephemeral-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-data-index-postgresql-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-data-index-ephemeral-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-db-migrator-tool-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-swf-builder-rhel8:1.36.0
  - name: registry.redhat.io/openshift-serverless-1/logic-swf-devmode-rhel8:1.36.0

  helm:
    repositories:
      - name: openshift-charts
        url: https://charts.openshift.io
        charts:
          - name: redhat-developer-hub
            version: "1.8.2"
          - name: redhat-developer-hub-orchestrator-infra
            version: "1.8.2"
  operators:
    - catalog: registry.redhat.io/redhat/redhat-operator-index:4.19
      # For example: registry.redhat.io/redhat/redhat-operator-index:v4.19
      packages:
      - name: logic-operator-rhel8
```

```
    channels:
    - name: alpha
      minVersion: 1.36.0
      maxVersion: 1.36.0
  - name: serverless-operator
    channels:
    - name: stable
      minVersion: 1.36.0
      maxVersion: 1.36.1
```

2. Mirror the images in the **ImageSetConfiguration.yaml** file by running the **oc-mirror** command to pull images and charts, and push the images directly to the target registry. For example:

```
$ oc-mirror --config=imagesetconfiguration.yaml docker://<registry URL:port> --workspace file://<workspace folder> --authfile /path/to/authfile  --v2
```

NOTE

The **oc-mirror** command pulls the charts listed in the **ImageSetConfiguration** file and makes them available as **tgz** archives under the **<workspace folder>** directory.

3. Apply the generated cluster resources to the disconnected cluster. For example:

```
$ cd <workspace folder>/working-dir/cluster-resources/
$ oc apply -f .
```

4. Download the Node Package Manager (NPM) packages for orchestrator 1.8.2 using any of the following methods:

- Download them as **tgz** files from the following registry:
  - https://npm.registry.redhat.com/@redhat/backstage-plugin-orchestrator/-/backstage-plugin-orchestrator-1.8.2.tgz
  - https://npm.registry.redhat.com/@redhat/backstage-plugin-orchestrator-backend-dynamic/-/backstage-plugin-orchestrator-backend-dynamic-1.8.2.tgz
  - https://npm.registry.redhat.com/@redhat/backstage-plugin-scaffolder-backend-module-orchestrator-dynamic/-/backstage-plugin-scaffolder-backend-module-orchestrator-dynamic-1.8.2.tgz
  - https://npm.registry.redhat.com/@redhat/backstage-plugin-orchestrator-form-widgets/-/backstage-plugin-orchestrator-form-widgets-1.8.2.tgz
- Alternatively, use the NPM packages from Red Hat NPM registry as shown in the following example:

```
$ npm pack "@redhat/backstage-plugin-orchestrator@1.8.2" --registry=https://npm.registry.redhat.com
$ npm pack "@redhat/backstage-plugin-orchestrator-backend-dynamic@1.8.2" --registry=https://npm.registry.redhat.com
$ npm pack "@redhat/backstage-plugin-scaffolder-backend-module-orchestrator-
```

```
dynamic@1.8.2 --registry=https://npm.registry.redhat.com
$ npm pack "@redhat/backstage-plugin-orchestrator-form-widgets@1.8.2" --
registry=https://npm.registry.redhat.com
```

5. Push the NPM packages you have downloaded to your NPM server, as shown in the following example:

```
$ npm publish backstage-plugin-orchestrator-1.8.2.tgz
$ npm publish backstage-plugin-orchestrator-backend-dynamic-1.8.2.tgz
$ npm publish backstage-plugin-orchestrator-form-widgets-1.8.2.tgz
$ npm publish backstage-plugin-scaffolder-backend-module-orchestrator-dynamic-1.8.2.tgz
```

6. Apply the **redhat-developer-hub-orchestrator-infra** Helm chart and approve the install plans. See Air-gapped installation with Helm chart instructions for details.

7. Apply the RHDH 1.8 Helm chart. Include the version 1.8.2 and enable the Orchestrator plugin, as shown in the following example:

```
orchestrator.enabled=true
```

8. The RHDH 1.8 Helm chart defaults to pulling Orchestrator plugins from the official Red Hat NPM registry using full URL references. You must override this behavior to point to your local registry. To configure the Orchestrator plugins to use a custom registry, complete the following steps:

   - Open your **values.yaml** file.

   - Explicitly list the Orchestrator plugin packages under the **orchestrator.plugins** section. You must replace the simplified package references with the full URLs that point to your custom NPM registry, as shown in the following example:

```
orchestrator:
  plugins:
    - disabled: false
      package: <custom_NPM_registry_URL>[:<port>]/@redhat/backstage-plugin-
orchestrator-backend-dynamic/-/backstage-plugin-orchestrator-backend-dynamic-
{product-bundle-version}.tgz
      integrity: sha512-xxxxxx
    - disabled: false
      package: <custom_NPM_registry_URL>[:<port>]/@redhat/backstage-plugin-
orchestrator/-/backstage-plugin-orchestrator-{product-bundle-version}.tgz
      integrity: sha512-xxxxxy
    - disabled: false
      package: <custom_NPM_registry_URL>[:<port>]/@redhat/backstage-plugin-
orchestrator-form-widgets/-/backstage-plugin-orchestrator-form-widgets-{product-bundle-
version}.tgz
      integrity: sha512-xxxxxz
    - disabled: false
      package: <custom_NPM_registry_URL>[:<port>]/@redhat/backstage-plugin-
scaffolder-backend-module-orchestrator-dynamic/-/backstage-plugin-scaffolder-backend-
module-orchestrator-dynamic-{product-bundle-version}.tgz
      integrity: sha512-xxxx1
```

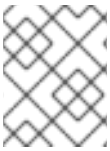
   where:

   <custom_NPM_registry_URL>

Enter the address of your custom registry and make sure the integrity checksum, such as sha512-xxxxxx, matches the files in your registry.

Verification

- Restart the RHDH pod and wait for the components to deploy properly.
- After deployment is complete, go to the **RHDH** UI and confirm that the Orchestrator UI is accessible and functioning correctly.



NOTE

The successful accessibility of the Orchestrator UI confirms that the underlying components are running and the cluster recognizes the plugin.

CHAPTER 7. BUILD AND DEPLOY SERVERLESS WORKFLOWS

To deploy a workflow and make it available in the Orchestrator plugin, follow these main steps:

- Building workflow images
- Generating workflow manifests
- Deploying workflows to a cluster

This process moves the workflow from your local machine to deployment on a cluster.

7.1. BENEFITS OF WORKFLOW IMAGES

While the OpenShift Serverless Logic Operator supports the building of workflows dynamically, this approach is primarily for experimentation. For production deployments, building images is the preferred method due to the following reasons:

- Production readiness: Prebuilt images can be scanned, secured, and tested before going live.
- GitOps compatibility: The Orchestrator relies on a central OpenShift Serverless Logic Operator instance to track workflows and their state. To use this tracking service, you must deploy workflows with the **gitops** profile, which expects a prebuilt image.
- Testing and quality: Building an image gives you more control over the testing process.

7.1.1. Project structure overview

The project utilizes **Quarkus project layout** (Maven project structure). This structure is illustrated by the following **01_basic** workflow example:

```
01_basic
├── pom.xml
├── README.md
└── src
    └── main
        ├── docker
        │   ├── Dockerfile.jvm
        │   ├── Dockerfile.legacy-jar
        │   ├── Dockerfile.native
        │   └── Dockerfile.native-micro
        └── resources
            ├── application.properties
            ├── basic.svg
            ├── basic.sw.yaml
            ├── schemas
            │   ├── basic__main-schema.json
            │   └── workflow-output-schema.json
            └── secret.properties
```

The main workflow resources are located under the **src/main/resources/** directory.

The **kn-workflow CLI** generated this project structure. You can try generating the structure yourself by following the *Getting Started guide*. For more information on the Quarkus project, see Creating your first application.

7.1.2. Creating and running your serverless workflow project locally

The **kn-workflow** CLI is an essential tool that generates workflow manifests and project structures. To ensure successful development and immediate testing, begin developing a new serverless workflow locally by completing the following steps:

Procedure

1. Use the **kn-workflow** CLI to create a new workflow project, which adheres to the Quarkus structure as shown in the following example:

```
$ kn-workflow quarkus create --name <specify project name, for example ,00_new_project>
```

2. Edit the workflow, add schema and specific files, and run it locally from project folder as shown in the following example:

```
$ kn-workflow quarkus run
```

3. Run the workflow locally using the **kn-workflow run** which pulls the following image:

```
registry.redhat.io/openshift-serverless-1/logic-swf-devmode-rhel8:1.36.0
```

4. For building the workflow image, the **kn-workflow** CLI pulls the following images:

```
registry.redhat.io/openshift-serverless-1/logic-swf-builder-rhel8:1.36.0-8
registry.access.redhat.com/ubi9/openjdk-17:1.21-2
```

Additional resources

- About OpenShift Serverless Logic
- OpenShift Serverless Logic Tutorial
- Running Red Hat Developer Hub behind a corporate proxy
- Using the Red Hat-hosted Quarkus repository

7.2. BUILDING WORKFLOW IMAGES LOCALLY

You can use the build script (**build.sh**) to build workflow images. You can run it either locally or inside a container. This section highlights how build workflow images locally.

Procedure

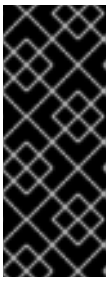
1. Clone the project as shown in the following example:

```
git clone git@github.com:rhdhorchestrator/orchestrator-demo.git
cd orchestrator-demo
```

2. Check the help menu of the script:

```
./scripts/build.sh --help
```

3. Run the **build.sh** script, providing the required flags, for instance, the image path (**-i**), workflow source directory (**-w**), and manifests output directory (**-m**).



IMPORTANT

You must specify the full target image path with a tag as shown in the following example:

```
./scripts/build.sh --image=quay.io/orchestrator/demo-basic:test -w 01_basic/ -m 01_basic/manifests
```

7.2.1. The build-sh script functionality and important flags

The **build-sh** script does the following tasks in order:

- Generates workflow manifests using the **kn-workflow** CLI.
- Builds the workflow image using **podman** or **docker**.
- Optional: The script pushes the images to an image registry and deploys the workflow using **kubectl**.

You can review the script configuration options and see available flags and their functions by accessing the help menu:

```
./scripts/build.sh [flags]
```

The following flags are essential for running the script:

Flag	Description
-i, **--image**	Required: Full image path, for example, **quay.io/orchestrator/demo:latest**
-w, **--workflow-directory**	Workflow source directory (default is the current directory)
-m, **--manifests-directory**	Where to save generated manifests
--push	Push the image to the registry
--deploy	Deploy the workflow
-h, **--help**	Show the help message

TIP

The script also supports builder and runtime image overrides, namespace targeting, and persistence flags.

7.2.2. Environment variables supported by the build script

The **build-sh** script supports the following environment variables to customize the workflow build process without modifying the script itself:

QUARKUS_EXTENSIONS

The **QUARKUS_EXTENSIONS** variable specifies additional Quarkus extensions required by the workflow. This variable takes the format of a comma-separated list of fully qualified extension IDs as shown in the following example:

```
export QUARKUS_EXTENSIONS="io.quarkus:quarkus-smallrye-reactive-messaging-kafka"
```

Add Kafka messaging support or other integrations at build time.

MAVEN_ARGS_APPEND

The **MAVEN_ARGS_APPEND** variable appends additional arguments to the **Maven build** command. This variable takes the format of a string of Maven CLI arguments as shown in the following example:

```
export MAVEN_ARGS_APPEND="-DmaxYamlCodePoints=35000000"
```

Control build behavior. For example, set **maxYamlCodePoints** parameter that controls the maximum input size for YAML input files to 35000000 characters (~33MB in UTF-8).

7.2.3. Required tools

To run the **build-sh** script locally and manage the workflow lifecycle, you must install the following command-line tools:

Tool	Conceptual Purpose.
podman or docker	Container runtime required for building the workflow images.
kubectl	Kubernetes CLI.
yq	YAML processor.
jq	JSON processor.
curl, **git**, **find**, **which**	Shell utilities.
kn-workflow	CLI for generating workflow manifests.

7.2.4. Building the 01_basic workflow

To run the script from the root directory of the repository, you must use the **-w** flag to point to the workflow directory. Additionally, specify the output directory with the **-m** flag.

Prerequisites

Prerequisites

- You have specified the target image using a tag.

Procedure

1. Run the following command:

```
$ ./scripts/build.sh --image=quay.io/orchestrator/demo-basic:test -w 01_basic/ -m
01_basic/manifests
```

   This build command produces the following two artifacts:

   - A workflow image and Kubernetes manifests: **quay.io/orchestrator/demo-basic:test** and tagged as **latest**.
   - Kubernetes manifests under: **01_basic/manifests/**

2. Optional: You can add the **--push** flag to automatically push the image after building. Otherwise, pushing manually is mandatory before deploying.

7.3. GENERATED WORKFLOW MANIFESTS

The following example is an illustration of what is generated under the **01_basic/manifests**:

```
01_basic/manifests
 ├── 00-secret_basic-secrets.yaml
 ├── 01-configmap_basic-props.yaml
 ├── 02-configmap_01-basic-resources-schemas.yaml
 └── 03-sonataflow_basic.yaml
```

00-secret_basic-secrets.yaml

Contains secrets from **01_basic/src/main/resources/secret.properties**. Values are not required at this stage as you can set them later after applying CRs or when using GitOps.

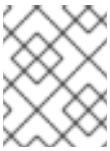
In OpenShift Serverless Logic **v1.36**, after updating a secret, you must manually restart the workflow Pod for changes to apply.

01-configmap_basic-props.yaml

Holds application properties from application.properties. Any change to this ConfigMap triggers an automatic Pod restart.

02-configmap_01-basic-resources-schemas.yaml

Contains JSON schemas from src/main/resources/schemas.



NOTE

You do not need to deploy certain configuration resources when using the GitOps profile.

03-sonataflow_basic.yaml

The SonataFlow custom resource (CR) that defines the workflow.

```
podTemplate:
  container:
    image: quay.io/orchestrator/demo-basic
    resources: {}
    envFrom:
      - secretRef:
          name: basic-secrets
```

```
persistence:
  postgresql:
    secretRef:
      name: sonataflow-psql-postgresql
      userKey: <your_postgres_username>
      passwordKey: <your_postgres_password>
    serviceRef:
      name: sonataflow-psql-postgresql
      port: 5432
      databaseName: sonataflow
      databaseSchema: basic
```

where:

postgresql:secretRef:name

Enter the Secret name for your deployment.

postgresql:secretRef:userKey

Enter the key for your deployment.

postgresql:secretRef:passwordKey

Enter the password for your deployment.

postgresql:serviceRef:name

Enter the Service name for your deployment.
If you must connect to an external database, replace **serviceRef** with **jdbcUrl**. See Managing workflow persistence.

By default, the script generates all the manifests without a namespace. You can specify a namespace to the script by using the **--namespace** flag if you know the target namespace in advance. Otherwise, you must provide the namespace when applying the manifests to the cluster. See Configuring workflow services.

7.4. DEPLOYING WORKFLOWS ON A CLUSTER

You can deploy the workflow on a cluster, because the image is pushed to the image registry and the deployment manifests are available.

Prerequisites

- You have an OpenShift Container Platform cluster with the following versions of components installed:
  - Red Hat Developer Hub (RHDH) 1.8
  - Orchestrator plugins 1.8.2

- OpenShift Serverless **v1.36**
- OpenShift Serverless Logic **v1.36**
  For instructions on how to install these components, see the Orchestrator plugin components on OpenShift Container Platform.

- You must apply the workflow manifests in a namespace that contains a **SonataflowPlatform** custom resource (CR), which manages the supporting services.

Procedure

1. Use the **kubectl create** command specifying the target namespace to apply the Kubernetes manifests as shown in the following example:

```
$ kubectl create -n <your_namespace> -f ./01_basic/manifests/.
```

2. After deployment, monitor the status of the workflow pods as shown in the following example:

```
$ kubectl get pods -n <your_namespace> -l app=basic
```

   The pod may initially appear in an **Error** state because of missing or incomplete configuration in the Secret or ConfigMap.

3. Inspect the Pod logs as shown in the following example:

```
$ oc logs -n <your_namespace> basic-f7c6ff455-vwl56
```

   The following code is an example of the output:

```
SRCFG00040: The config property quarkus.openapi-
generator.notifications.auth.BearerToken.bearer-token is defined as the empty String ("")
which the following Converter considered to be null:
io.smallrye.config.Converters$BuiltInConverter
java.lang.RuntimeException: Failed to start quarkus
...
Caused by: io.quarkus.runtime.configuration.ConfigurationException: Failed to read
configuration properties
```

   The error indicates a missing property: **quarkus.openapi-generator.notifications.auth.BearerToken.bearer-token**.

4. In such a case where the logs show the **ConfigurationException: Failed to read configuration properties** error or indicate a missing value, retrieve the ConfigMap as shown in the following example:

```
$ oc get -n <your_namespace> configmaps basic-props -o yaml
```

   The following code is an example of the sample output:

```
apiVersion: v1
data:
  application.properties: |
    # Backstage notifications service
    quarkus.rest-client.notifications.url=${BACKSTAGE_NOTIFICATIONS_URL}
```

```
    quarkus.openapi-generator.notifications.auth.BearerToken.bearer-
token=${NOTIFICATIONS_BEARER_TOKEN}
...
```

Resolve the placeholders using values provided using a Secret.

5. You must edit the corresponding Secret and provide appropriate base64-encoded values to resolve the placeholders in **application.properties** as shown in the following example:

```
$ kubectl edit secrets -n <your_namespace> basic-secrets
```

6. Restart the workflow Pod for Secret changes to take effect in OpenShift Serverless Logic **v1.36**.

Verification

1. Verify the deployment status by checking the Pods again as shown in the following example:

```
$ oc get pods -n <your_namespace> -l app=basic
```

The expected status for a successfully deployed workflow Pod is as shown in the following example:

```
NAME                READY   STATUS    RESTARTS   AGE
basic-f7c6ff455-grkxd   1/1     Running   0          47s
```

2. Once the Pod is in the **Running** state, the workflow now appears in the Orchestrator plugin inside the Red Hat Developer Hub.

Next steps

- Inspect the provided build script to extract the actual steps and implement them in your preferred CI/CD tool, for example, GitHub Actions, GitLab CI, Jenkins, and Tekton.

7.5. BEST PRACTICES WHEN CREATING SERVERLESS WORKFLOWS

Create effective serverless workflows using thoughtful approaches to design, handle data, and manage error by following these best practices based on the Serverless Workflow Domain Specific Language (DSL) principles. These principles help you to build robust workflows.

Workflow design principles

The Serverless Workflow DSL prioritizes clarity and ease of use when writing workflows.

Priority of constituencies

When developing workflows or APIs, ensure the needs of the author (workflow writer) come first. The constituencies are prioritized in the following order: Authors > Operators > Implementors > Specifications writers.

Linguistic fluency and clarity

- Use imperative verbs such as **Call**, **Emit**, **For**, **Fork**, **Raise**, **Run**, **Set**, **Switch**, and **Wait**. These simple, universally understood terms make your workflow simple to read and understand.

Structure and extensibility

- Use implicit default behaviors to reduce redundancy.
- Declare components inline if they are not reusable to keep the definition self-contained.
- Use external references to import and reuse shared components, which promotes a modular design.
- Prioritize flexibility over strict enumerations to ensure extensibility and adaptability across different runtime environments.

Data flow and runtime management

Controlling data flow is critical for efficient workflows. Tasks are the fundamental computing units of a workflow. The Domain Specific Language (DSL) defines several default task types that runtimes must do. These include **Do**, **Listen**, **Raise**, **Run**, **Try**, and **Wait**.

Security and error handling

Secrets

Use Secrets with caution. Avoid passing them directly in call inputs as this might expose sensitive information.

Fault tolerance and error handling

Serverless Workflow is designed with resilience in mind to recover from failures.

Orchestrator UI integration best practices

For your workflow results to be effectively displayed in the Orchestrator UI and to facilitate chaining of workflows, you must structure the output data according to the **WorkflowResult** schema. Additionally, include any error information as part of the workflow output so the UI and subsequent workflows can handle them accordingly.

Workflow output schema

Results placement

The primary output intended for subsequent processing must be placed under the **data.result** property.

Schema reference

Your output schema file (**schemas/workflow-output-schema.json**) must reference the **WorkflowResult** schema.

Outputs definition

Include an **outputs** section in your workflow definition. This section contains human-readable key/value pairs that the UI will display.
Structure of workflow:

```
id: my-workflow
version: "0.8"
specVersion: "0.8"
name: My Workflow
start: ImmediatelyEnd
dataInputSchema: schemas/basic__main-schema.json
extensions:
  - extensionid: workflow-output-schema
    outputSchema: schemas/workflow-output-schema.json
```

```
functions:
  - name: print
    type: custom
    operation: sysout
  - name: successResult
    type: expression
    operation: '{
      "result": {
      "message": "Project " + .projectName + " active",
      "outputs":[]
      }
      }'
start: "successResult"
states:
  - name: successResult
    type: operation
    actions:
      - name: setOutput
        functionRef:
          refName: successResult
    end: true
```

CHAPTER 8. AUTOMATING WORKFLOW DEPLOYMENT WITH ORCHESTRATOR

The Orchestrator plugin for Red Hat Developer Hub (RHDH) automates the software development lifecycle (SDLC) for serverless workflows. By using the Orchestrator software templates, you bootstrap a complete serverless workflow project that includes a fully operational Git repository, deployment configurations, and automated continuous integration (CI) and continuous deployment (CD) pipelines.

NOTE

Use the **rhdh** namespace where the RHDH chart is installed.

The Orchestrator plugin integrates these components:

RHDH Helm chart

Installs the RHDH Orchestrator.

Tekton or Red Hat OpenShift Pipelines

Manages the **Kubernetes-native** CI pipeline to build images.

ArgoCD or Red Hat OpenShift GitOps

Manages the CD pipeline to deploy the workflow on the RHDH instance.

Quay.io

Stores the container images generated by the pipelines.

OpenShift Serverless Logic operator

Implements serverless workflow specifications

8.1. INSTALLING ORCHESTRATOR SOFTWARE TEMPLATES

To enable software templates on RHDH, you must install two additional Helm charts.

Prerequisites

- You have installed RHDH and the Orchestrator plugin by using the Helm chart.
- You have installed the **redhat-developer-hub-orchestrator-infra** chart.

Procedure

1. Install the **orchestrator-software-templates-infra** chart.
2. Install the **orchestrator-software-templates** chart.

8.1.1. Installing the Orchestrator Software Templates Infra chart

The **orchestrator-software-templates-infra** chart installs the Custom Resource Definitions (CRDs) and operators for Tekton (Red Hat OpenShift Pipelines) and Argo CD (Red Hat OpenShift GitOps). These are required to handle the CI/CD automation for serverless workflows.

Prerequisites

- You have **cluster-admin** privileges.
- You have installed the Helm CLI.
- You have added the following plugins to the RHDH chart **values.yaml** file to include the following dynamic plugins:
  - **backstage-plugin-scaffolder-backend-module-github-dynamic**
  - **backstage-plugin-scaffolder-backend-module-gitlab-dynamic**
  - **backstage-plugin-kubernetes-backend-dynamic**
  - **backstage-plugin-kubernetes**
  - **backstage-community-plugin-tekton**
  - **backstage-community-plugin-redhat-argocd**
  - **roadiehq-backstage-plugin-argo-cd-backend-dynamic**
  - **roadiehq-scaffolder-backend-argocd-dynamic**
    Edit the **values.yaml** and upgrade the chart.

Procedure

- Install the infrastructure chart:

```
$ helm install <release_name> redhat-developer/redhat-developer-hub-orchestrator-infra
```

Verification

- Verify the installation by running the following command:

```
$ helm test redhat-developer-hub-orchestrator-infra
```

8.1.2. Installing the Orchestrator Software Templates chart

The **orchestrator-software-templates** chart loads the actual software templates into your RHDH instance. This allows users to select workflow templates from the RHDH Catalog.

Prerequisites

- You have installed the **orchestrator-software-templates-infra** chart to deploy OpenShift Pipelines (Tekton) operator and OpenShift GitOps (ArgoCD) operator in the same namespace as RHDH.
- You have labeled the **rhdh** namespace to enable GitOps sync:

```
$ oc label ns rhdh rhdh.redhat.com/argocd-namespace=true
```

- You have created a secret named **orchestrator-auth-secret** in the **rhdh** namespace containing the following keys:

- **BACKEND_SECRET**: Backend authentication secret
- **K8S_CLUSTER_TOKEN**: Kubernetes cluster token
- **K8S_CLUSTER_URL**: Kubernetes cluster URL
- **GITHUB_TOKEN**: GitHub access token (optional)
- **GITHUB_CLIENT_ID**: GitHub OAuth client ID (optional)
- **GITHUB_CLIENT_SECRET**: GitHub OAuth client secret (optional)
- **GITLAB_HOST**: GitLab host URL (optional)
- **GITLAB_TOKEN**: GitLab access token (optional)
- **ARGOCD_URL**: ArgoCD server URL (optional)
- **ARGOCD_USERNAME**: ArgoCD username (optional)
- **ARGOCD_PASSWORD**: ArgoCD password (optional)

Procedure

1. Install the software templates chart:

```
$ helm repo add redhat-developer https://redhat-developer.github.io/rhdh-chart
$ helm install my-orchestrator-templates redhat-developer/orchestrator-software-templates --
version 0.2.0
```

2. Create your environment-specific values file:
   a. Retrieve your RHDH route URL:

```
RHDH_ROUTE="https://$(oc get route -n {{
.Values.orchestratorTemplates.rhdhChartNamespace }} -o
jsonpath='{.items[0].spec.host}')"
```

   b. Copy the template and replace placeholders

```
cp charts/orchestrator-software-templates/orchestrator-templates-values.yaml.template
orchestrator-templates-values.yaml
sed -i "s|__RHDH_BASE_URL__|$RHDH_ROUTE|g" orchestrator-templates-
values.yaml
```

3. Backup your RHDH configuration:

```
helm show values charts/backstage \
    -n {{ .Values.orchestratorTemplates.rhdhChartNamespace }} > current-backstage-
values.yaml
```

4. Upgrade the RHDH chart with both value files:

```
helm upgrade {{ .Values.orchestratorTemplates.rhdhChartReleaseName }} charts/backstage
```

```
\
    -n {{ .Values.orchestratorTemplates.rhdhChartNamespace }} \
    -f current-backstage-values.yaml \
    -f orchestrator-templates-values.yaml
```

Verification

1. Wait for the deployment to complete.
2. Open your RHDH instance and verify the new software templates appear in the **Create** menu.

8.2. CREATING A SERVERLESS WORKFLOW PROJECT

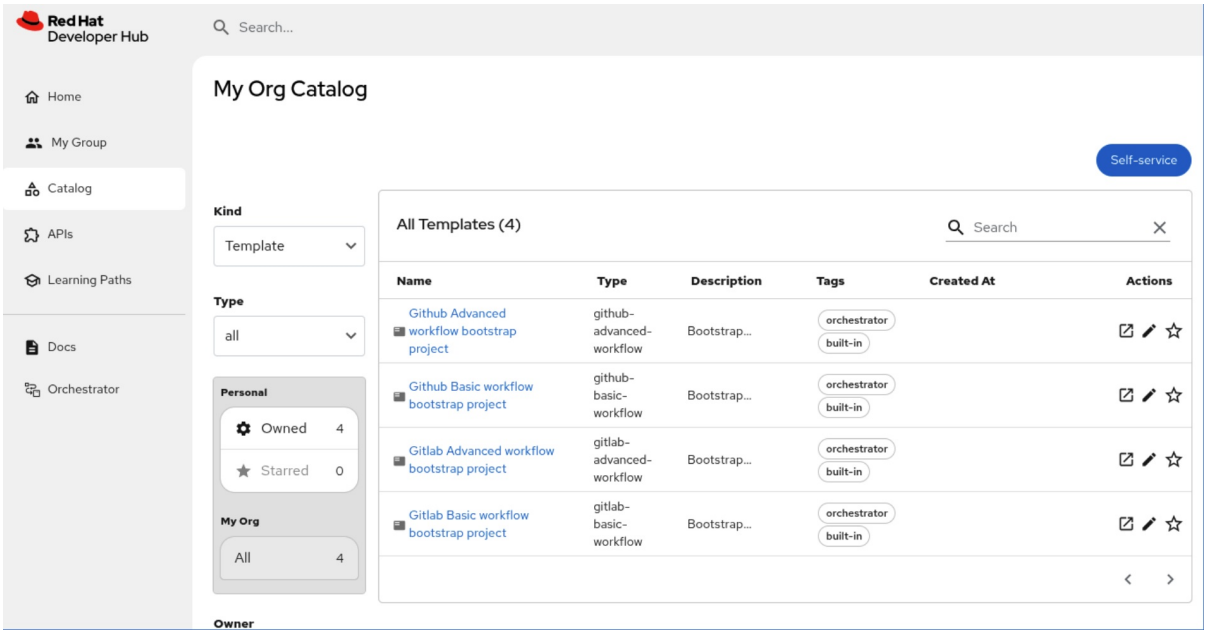
Use the Orchestrator software templates to generate a project that includes workflow definitions, **Kustomize** configurations, and CI/CD pipelines.

Prerequisites

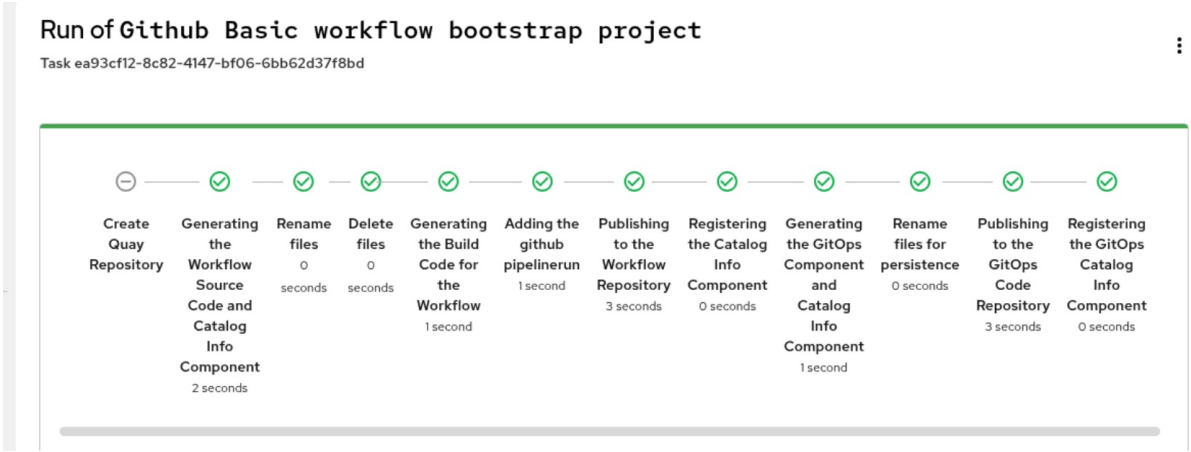
- You have installed **orchestrator-software-templates-infra** and **orchestrator-software-templates** Helm charts to enable templates.
- You have installed RHDH and the Orchestrator plugin by using the Helm chart.
- You have a **Quay.io** organization and repository for storing the workflow images.
- You have a GitHub or Gitlab personal access token with repository creation permissions.
- You have configured a GitOps secret for the target cluster.
- You have set the target namespace for both the pipeline and the workflow to the **rhdh** namespace.

Procedure

1. Prepare the image registry. Before creating the template, configure the target repository in **Quay.io**.
    a. Log in to your **Quay.io** organization (for example, **orchestrator-testing**).
    b. Create a new repository (for example, **serverless-workflow-demo**).
    c. Add robot account permissions to the repository settings.
2. Open the Red Hat Developer Hub Catalog.



3. Select the **Basic workflow bootstrap project** template and click **Launch Template**.
4. Follow the template form to enter required details, including the **GitHub organization**, **source code repository name**, and a unique **Workflow ID**.
5. For the CI/CD method, select **Tekton with Argo CD** to generate GitOps resources.
6. Set the **Workflow Namespace** to **rhdh** and the **GitOps Namespace** to **orchestrator-gitops**.
7. Enter your **Quay.io** registry details.
8. Click **Review**, then click **Create**.



9. Optional: Enable persistence and provide database connection details if the workflow requires a database schema.

Verification

- The system creates the following repositories:
  - Source code repository: Contains the serverless workflow project.
  - GitOps repository: Contains GitOps configurations, Tekton pipeline templates, and bootstrap instructions.

Additional resources

- Building and deploying serverless workflows

8.3. BOOTSTRAPPING GITOPS RESOURCES AND TRIGGERING PIPELINES

You must manually bootstrap the GitOps resources to trigger the continuous integration (CI) pipeline.

Procedure

1. Open the generated **GitOps repository**.
2. Clone the repository and navigate to the **bootstrap** directory:

```
$ git clone https://token:<PAT>@${{ values.gitHost }}/${{ values.orgName }}/${{
values.repoName }}.git
cd <repo_name>/bootstrap
```

NOTE

If you are not authenticated, you must use a personal access token (PAT) in the clone URL. Make sure the PAT has repository access permissions.

3. Open **${{values.workflowId}}-argocd-repo.yaml** and replace the ***REPLACE_SSH_PRIVATE_KEY*** string with your SSH private key.
4. Apply the manifests to the cluster:

```
$ kubectl apply -f .
```

Applying these manifests triggers the following automated sequence:

a. CI Pipeline (Tekton): Builds the workflow image and pushes it to your **Quay.io** registry.

b. CD Pipeline (Argo CD): Deploys the serverless workflow manifests to the cluster.

8.4. VERIFYING THE DEPLOYMENT

Verify the status of your continuous integration (CI) and continuous deployment (CD) pipelines in the RHDH component catalog.

Procedure

1. For CI:

a. In the RHDH Catalog, select your source code repository component (for example, **onboardings**).

b. Click the **CI** tab and verify that the pipeline run status is **Succeeded**.

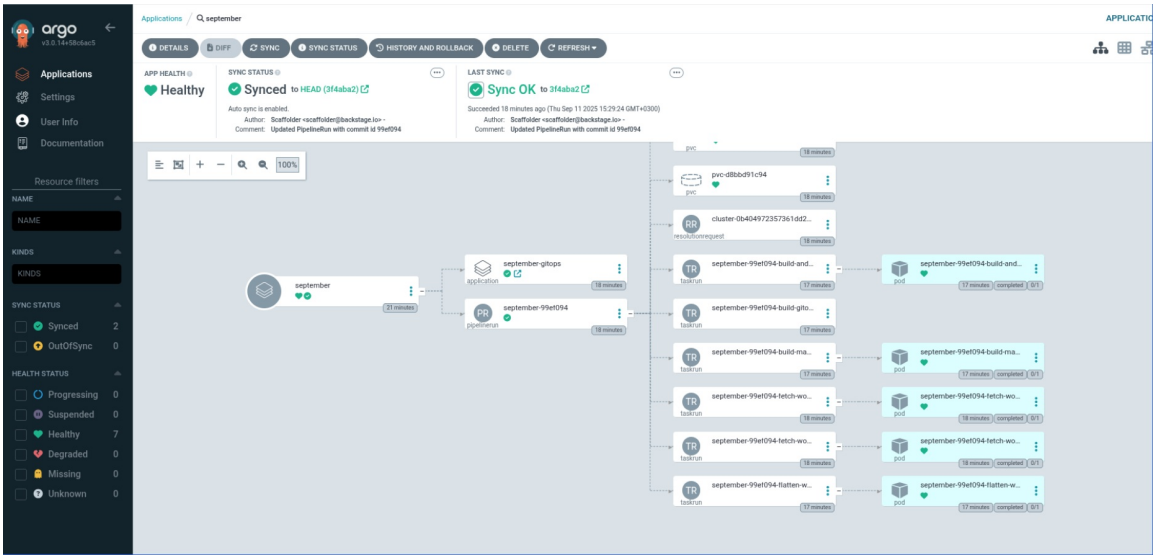
c. If the pipeline status does not appear in the Red Hat Developer Hub console, verify the CI status directly in your Git provider (GitHub or GitLab).

d. If the pipeline fails, click the **run** name to view the logs and identify build errors.

2. For CD:

a. Open the GitOps Resources Repository component in the Catalog (for example, **onboarding-gitops**).

b. Click the **CD** tab and make sure the Kubernetes resources are **synced** and **healthy**. This confirms that ArgoCD deployed the workflow to the cluster.



8.5. TROUBLESHOOTING WORKFLOW DEPLOYMENTS

Identify and resolve issues related to plugin visibility, pipeline execution, or resource synchronization.

1. Visibility issues

Missing Orchestrator plugin

If Orchestrator features do not appear in RHDH, make sure you have updated the RHDH Helm chart with the required plugins.

Software templates not appearing

Make sure the **orchestrator-software-templates** chart is installed and the **orchestrator-auth-secret** exists in the correct namespace.

2. Pipeline failure (CI)

GitHub or GitLab actions failure

The GitOps automation includes a GitHub Action or GitLab CI step that creates a **PipelineRun** manifest from a **PipelineRun** template. Examine the failed GitHub or GitLab actions logs. Failures often occur due to invalid Git credentials or misconfigured runner permissions. You can also create the **PipelineRun** file manually to bypass automation issues.

Build or push issues

Check the **CI** tab in the RHDH Catalog.
If RHDH does not display the status, use the OpenShift Container Platform console to monitor pipeline instances and triggered jobs. Navigate to **Pipelines** > **PipelineRuns** for detailed logs.

If the Tekton pipeline fails during the build or push stages:

- Verify that your **Quay.io** robot account has **Write** permissions.
- Ensure the **docker-registry-credentials** secret exists in the **rhdh** namespace.

3. Resource visibility and Sync issues (CD)

Pipeline succeeds but workflows are missing

If the CI pipeline succeeds but the workflow does not appear in the **CD** tab:

- Make sure the target namespace is labeled for Argo CD:

```
$ oc label ns sonataflow-infra rhdh.redhat.com/argocd-namespace=true
```

- Make sure the ArgoCD ServiceAccount has the required permissions to manage resources in the **rhdh** namespace.

Argo CD sync failure

If resources appear but remain in an **OutOfSync** state, click **Refresh** in the Argo CD UI or verify that the **AppProject** exists in the **orchestrator-gitops** namespace.

CHAPTER 9. DIAGNOSE AND RESOLVE SERVERLESS WORKFLOW ISSUES

Use the following information to diagnose and resolve serverless workflow and visibility issues.

9.1. TROUBLESHOOT WORKFLOW HTTP ERROR CODES

Workflow operations fail when a service endpoint returns an HTTP error code. The user interface displays the HTTP code and error message. See external documentation for a complete list of HTTP status code meanings.

The following table lists common HTTP errors encountered during workflow execution:

HTTP code	Description	Possible cause
401	Unauthorized access	The token, password, or username provided for the endpoint might be incorrect or expired.
403	Forbidden	The server understood the request but refused to process it due to insufficient permissions to a resource or action.
409	Conflict	The workflow attempted to create or update a resource (for example, Kubernetes or OpenShift resources) that already exists.

9.2. TROUBLESHOOTING COMMON WORKFLOW DEPLOYMENT ERRORS

Use these steps to diagnose and resolve common workflow deployment, connectivity, or configuration failures.

Procedure

1. If the workflow operation fails, examine the container log of the specific workflow instance to determine the cause by running the following command:

```
$ oc logs my-workflow-xy73lj
```

2. If the workflow fails to reach an HTTPS endpoint, check the pod log for an SSL certificate verification failure. This occurs if the target endpoint uses a Certificate Authority (CA) that the workflow cannot verify. The resulting error resembles the following:

```
sun.security.provider.certpath.SunCertPathBuilderException - unable to find valid certification path to requested target
```

3. To resolve the SSL certificate error, load the additional CA certificate into the running workflow container.

9.3. TROUBLESHOOTING CROSS-NAMESPACE SONATAFLOW CONFIGURATION AND DEPLOYMENT ISSUES

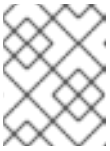
Use this procedure to resolve configuration and deployment failures when SonataFlow workflows are installed in a namespace separate from the core services, or if the Data Index fails to connect to the PostgreSQL database.

Prerequisites

- You have administrator privileges to access the OpenShift cluster.

Procedure

1. Identify required namespaces.
   - Retrieve the namespace value where RHDH is running using **oc get backstage -A**.
   - Identify the SonataFlow Services Namespace by checking for either a **sonataflowclusterplatform** or **sonataflowplatform** instance.



NOTE

By default, the SonataFlow namespace must be the same as the RHDH namespace.

2. If the workflow is deployed to a namespace outside the core SonataFlow services, configure network policies to permit the necessary inter-namespace traffic.

```
# Example NetworkPolicy configuration to ingress traffic into the workflow namespace
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: {{ .Release.Name }}-allow-infra-ns-to-workflow-ns
  # Sonataflow and Workflows are using the RHDH target namespace.
  namespace: {{ .Release.Namespace | quote }}
spec:
  podSelector: {}
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            # Allow knative events to be delivered to workflows.
            kubernetes.io/metadata.name: knative-eventing
      - namespaceSelector:
          matchLabels:
            # Allow auxiliary knative function for workflow (such as m2k-save-transformation)
            kubernetes.io/metadata.name: knative-serving
      - namespaceSelector:
          matchLabels:
            # Allow communication between the serverless logic operator and the workflow
namespace.
            kubernetes.io/metadata.name: openshift-serverless-logic
```

3. Add **SonataFlowClusterPlatform** Custom Resource as shown in the following configuration:

```
oc create -f - <<EOF
apiVersion: sonataflow.org/v1alpha08
kind: SonataFlowClusterPlatform
metadata:
  name: cluster-platform
spec:
  platformRef:
    name: sonataflow-platform
    namespace: $RHDH_NAMESPACE
```

4. To allow communication between RHDH namespace and the workflow namespace, create the following network policies:

   a. Allow RHDH services to accept traffic from workflows. Create an additional network policy within the RHDH instance namespace as shown in the following configuration::

```
oc create -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-external-workflows-to-rhdh
  # Namespace where network policies are deployed
  namespace: $RHDH_NAMESPACE
spec:
  podSelector: {}
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            # Allow SonataFlow services to communicate with new/additional workflow
namespace.
            kubernetes.io/metadata.name: $ADDITIONAL_WORKFLOW_NAMESPACE
```

   b. Allow traffic from RHDH, SonataFlow and Knative. Create a network policy within the additional workflow namespace as shown in the following configuration:

```
oc create -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-rhdh-and-knative-to-workflows
  namespace: $ADDITIONAL_WORKFLOW_NAMESPACE
spec:
  podSelector: {}
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            # Allows traffic from pods in the {product-very-short} namespace.
            kubernetes.io/metadata.name: $RHDH_NAMESPACE
      - namespaceSelector:
          matchLabels:
            # Allows traffic from pods in the Knative Eventing namespace.
            kubernetes.io/metadata.name: knative-eventing
      - namespaceSelector:
```

```
    matchLabels:
      # Allows traffic from pods in the Knative Serving namespace.
      kubernetes.io/metadata.name: knative-serving
```

5. (Optional) Create an **allow-intra-namespace** policy in the workflow namespace to enable unrestricted communication among all pods within that namespace.

6. If workflow persistence is required, perform the following configuration steps:

   a. Create a dedicated PostgreSQL Secret containing database credentials within the workflow namespace as shown in the following configuration:

   ```
   oc get secret sonataflow-psql-postgresql -n <your_namespace> -o yaml > secret.yaml
   sed -i '/namespace: <your_namespace>/d' secret.yaml
   oc apply -f secret.yaml -n $ADDITIONAL_NAMESPACE
   ```

   b. Configure the workflow **serviceRef** property to correctly reference the PostgreSQL service namespace as shown in the following configuration:

   ```
   apiVersion: sonataflow.org/v1alpha08
   kind: SonataFlow
     ...
   spec:
     ...
     persistence:
       postgresql:
         secretRef:
           name: sonataflow-psql-postgresql
           passwordKey: postgres-password
           userKey: postgres-username
         serviceRef:
           databaseName: sonataflow
           databaseSchema: greeting
           name: sonataflow-psql-postgresql
           namespace: $POSTGRESQL_NAMESPACE
           port: 5432
   ```

   namespace

   Enter the namespace where the PostgreSQL server is deployed.

7. If the **sonataflow-platform-data-index-service** cannot connect to the PostgreSQL database on startup, perform the following diagnostic checks:

   a. Verify that the PostgreSQL Pod has fully transitioned to a **running** and operational status. Allow additional time for database initialization before expecting related service pods (**DataIndex**, **JobService**) to establish a connection.

   b. If the PostgreSQL Server operates in a dedicated namespace (for example, outside RHDH), verify that network policies are configured to allow ingress traffic from the SonataFlow services namespace. Network policies might prevent the Data Index and Job Service pods from connecting to the database.

9.4. TROUBLESHOOTING WORKFLOWS MISSING FROM THE RHDH UI

You can perform the following checks to verify the workflow status and connectivity when the deployed workflow is missing from the RHDH Orchestrator UI.

Prerequisites

- You have administrator privileges to access the OpenShift cluster where RHDH and SonataFlow services are running.

Procedure

1. Verify if the workflow uses GitOps profile. The RHDH Orchestrator UI displays only the workflows that use this profile. Make sure the workflow definition and the SonataFlow manifests use the GitOps profile.

2. Verify that the workflow pod has started and is ready. The readiness of a workflow pod depends on its successful registration with the Data Index. When a workflow initializes, it performs the following actions:

   a. It attempts to create its schema in the database (if persistence is active).

   b. It attempts to register itself to the Data Index. The workflow pod remains in an unready state until it successfully registers to the Data Index.
   Check the workflow deployment for additional status and error messages that might be unavailable in the pod log.

3. Check if the workflow pod can reach the Data Index service. Connect to the workflows pod and send the following GraphQL request to the Data Index:

   ```
   curl -g -k  -X POST  -H "Content-Type: application/json" \
                   -d '{"query":"query{ ProcessDefinitions  { id, serviceUrl, endpoint } }"}' \
                   http://sonataflow-platform-data-index-service.<your_namespace>/graphql
   ```

   Use the Data Index service and namespace as defined in your environment. By default, this is the same namespace where RHDH is installed. If your SonataFlow resources are installed in a separate namespace, use **<your_namespace>**. Check if the RHDH pod can reach the workflow service by running the following command:

   ```
   curl http://<workflow_service>.<workflow_namespace>/management/processes
   ```

4. Connect to the RHDH pod. Verify its connection to the Data Index service and inspect the RHDH pod logs for messages from the Orchestrator plugin.
   To inspect the logs, identify the RHDH pod and run the following **oc logs** command:

   ```
   oc get pods -n <your_namespace>
   oc logs <rhdh_pod_name> -n <your_namespace>
   ```

   You must find messages indicating it is attempting to fetch workflow information from the Data Index, similar to the following:

   ```
   {"level":"\u001b[32minfo\u001b[39m","message":"fetchWorkflowInfos() called:
   http://sonataflow-platform-data-index-service.
   <your_namespace>","plugin":"orchestrator","service":"backstage","span_id":"fca4ab29f0a7aef
   9","timestamp":"2025-08-04
   17:58:26","trace_flags":"01","trace_id":"5408d4b06373ff8fb34769083ef771dd"}
   ```

Notice the *"plugin":"orchestrator"* that can help to filter the messages.

5. Make sure the Data Index properties are set in the **-managed-props** ConfigMap of the workflow as shown in the following configuration:

```
kogito.data-index.health-enabled = true
kogito.data-index.url = http://sonataflow-platform-data-index-service.<your_namespace>
...
mp.messaging.outgoing.kogito-processdefinitions-events.url = http://sonataflow-platform-data-index-service.<your_namespace>/definitions
mp.messaging.outgoing.kogito-processinstances-events.url = http://sonataflow-platform-data-index-service.<your_namespace>/processes
```

NOTE

The **-managed-props** ConfigMap is located in the same namespace as the workflow and is generated by the Openshift Serverless Logic (OSL) Operator.

These properties, along with similar settings for the Job Services, indicate that the (OSL) Operator successfully registered the Data Index service.

6. Confirm that the workflow is registered in the Data Index database. Connect to the database used by the Data Index and run the following command from the PSQL instance pod:

```
PGPASSWORD=<psql password> psql -h localhost -p 5432 -U < user> -d sonataflow
```

Replace **<psql password>** and **<user>** with your database credentials.

Run the following SQL commands to query the registered workflow definitions:

```
sonataflow=# SET search_path TO "sonataflow-platform-data-index-service";
sonataflow=# select id, name from definitions;
```

You must see your workflows listed in the query results.

7. Make sure you have enabled Data Index and Job Service in the **SonataFlowPlatform** custom resource (CR) as shown in the following configuration:

```
services:
  dataIndex:
    enabled: true
  jobService:
    enabled: true
```

If you fail to enable the Data Index and the Job Services in the **SonataFlowPlatform** custom resource (CR), the Orchestrator plugin fails to fetch the available workflows.

NOTE

You can also manually edit the **SonataFlowPlatform** CR instance to trigger the re-creation of workflow-related manifests.

8. Set the RBAC permissions correctly. For more information, see RBAC documentation.

Additional resources

- Configuring the application log level by using the Red Hat Developer Hub Operator

CHAPTER 10. TECHNICAL APPENDIX

The following appendix provides technical information, and details on non-production tools, such as the RHDH helper script, which might be helpful for understanding setup options or quick testing.

10.1. INSTALLING COMPONENTS USING THE RHDH HELPER SCRIPT

You can use the RHDH helper script **plugin-infra.sh** to quickly install the OpenShift Serverless infrastructure and Openshift Serverless Logic infrastructure required by the Orchestrator plugin.

WARNING

Do not use **plugin-infra.sh** in production.

Procedure

1. Download the **plugin-infra.sh** script as shown in the following example:

```
$ curl -sSLO https://raw.githubusercontent.com/redhat-developer/rhdh-operator/refs/heads/release-1.8/config/profile/rhdh/plugin-infra/plugin-infra.sh # Specify the Red Hat Developer Hub version in the URL or use main
```

2. Run the script:

```
$ ./plugin-infra.sh
```