Red Hat

Red Hat Developer Hub 1.8

Accelerate AI development with Openshift AI Connector for Red Hat Developer Hub

Installing, configuring, and troubleshooting OpenShift AI Connector for Red Hat Developer Hub

Last Updated: 2026-01-22

Red Hat Developer Hub 1.8 Accelerate AI development with Openshift AI Connector for Red Hat Developer Hub

Installing, configuring, and troubleshooting OpenShift AI Connector for Red Hat Developer Hub

Abstract

As a developer, when you require access to centralized AI/ML services, you can integrate AI models and model servers from Red Hat OpenShift AI directly into the Red Hat Developer Hub (RHDH) Catalog, so that you can provide a single, consistent hub for discovering, managing, and consuming all components, accelerating time-to-market.

Table of Contents

CHAPTER 1. UNDERSTAND HOW AI ASSETS MAP TO THE RED HAT DEVELOPER HUB CATALOG

IMPORTANT

This section describes Developer Preview features in the OpenShift AI Connector for Red Hat Developer Hub plugin. Developer Preview features are not supported by Red Hat in any way and are not functionally complete or production-ready. Do not use Developer Preview features for production or business-critical workloads. Developer Preview features provide early access to functionality in advance of possible inclusion in a Red Hat product offering. Customers can use these features to test functionality and provide feedback during the development process. Developer Preview features might not have any documentation, are subject to change or removal at any time, and have received limited testing. Red Hat might provide ways to submit feedback on Developer Preview features without an associated SLA.

For more information about the support scope of Red Hat Developer Preview features, see Developer Preview Support Scope.

The OpenShift AI Connector for Red Hat Developer Hub (OpenShift AI Connector for RHDH) serves as a crucial link, enabling the discovery and accessibility of AI assets managed within the Red Hat OpenShift AI offering directly within your RHDH instance.

For more information on model registry components, see Overview of model registries and model catalog.

1.1. MODEL-TO-ENTITY MAPPING

Model-to-Entity mapping integrates with OpenShift AI Connector for RHDH, the model catalog, and KServe-based Model Deployments (InferenceServices). This integration automatically converts your AI/ML artifacts into familiar Backstage entities, simplifying management and providing a unified view of your available AI models to your developer teams.

This offering interfaces with the OpenShift AI Connector for RHDH, model catalog, and KServe-based Model Deployments (InferenceServices) to create familiar Backstage entities.

RHOAI Artifact	RHDH/Backstage Entity Kind	RHDH/Backstage Entity Type	Purpose
Model Server (InferenceService)	Component	**model-server**	Represents a running, accessible AI model endpoint. See Configuring your model-serving platform.
AI Model (Model Registry Version)	Resource	**ai-model**	Represents the specific AI model artifact, for example, **Llama-3-8B**.

RHOAI Artifact	RHDH/Backstage Entity Kind	RHDH/Backstage Entity Type	Purpose
Model Server API Details	API	**openapi** (Default)	Provides the OpenAPI/Swagger specification for the REST endpoint of the model. See Red Hat OpenShifT AI: API Tiers
Model Cards	TechDocs	N/A	Model cards from the RHOAI model catalog are associated with the Component and Resource entities. See Registering a model from the model catalog.

Once the OpenShift AI Connector for RHDH is installed and connected with RHOAI, the transfer of information commences automatically.

1.2. OUT-OF-THE-BOX AS ASSET DETAILS SYNCHED FROM RHOAI

The connector propagates the following key data:

- InferenceServices (Component type model-server):
  - URL of the OpenShift Route (if exposed).
  - URL of the Kubernetes Service.
  - Authentication requirement status.
- Model registry (Resource type **ai-model**):
  - Model description, artifact URIs, and author/owner information.
- Model catalog:
  - Links to the Model Card (as RHDH TechDocs).
  - Model license URL.

CHAPTER 2. SETTING UP OPENSHIFT AI CONNECTOR FOR RED HAT DEVELOPER HUB WITH RED HAT OPENSHIFT AI

The installation of the OpenShift AI Connector for Red Hat Developer Hub requires manual updates to RHDH-related Kubernetes resources.

RHOAI Prerequisites

- To import model cards from the model catalog into TechDocs, you must use RHOAI 2.25.

  NOTE

  If you upgraded to RHOAI 2.25 from an earlier version, you must manually enable the model catalog dashboard and model registry before you can import model cards.

- If you used the model catalog in earlier versions of RHOAI, TechDocs propagation does not work for any models you registered into the model registry while at those earlier versions; only models registered into model registry from a RHOAI 2.25 model catalog have their model cards transferred to RHDH as TechDocs.
- For the rest of the features, version 2.20 or later suffices. Enabling model registry and its associated dashboard allows for a user experience that more directly allows for customizing AI Model metadata. For best overall experience, RHOAI 2.25 is recommended.

For more details, see Enabling the model registry component.

Procedure

1. Configure RHOAI-related RBAC and credentials. A Kubernetes **ServiceAccount** and a **service-account-token** Secret are required for the connector to retrieve data from RHOAI. The following resources must be created, replacing namespace names (**ai-rhdh** for RHDH, **rhoai-model-registries** for RHOAI) as needed:

   - **ServiceAccount** (**rhdh-rhoai-connector**). For example:

     ```
     apiVersion: v1
     kind: ServiceAccount
     metadata:
       name: rhdh-rhoai-connector
       namespace: ai-rhdh
     ```

   - **ClusterRole** and **ClusterRoleBinding** (**rhdh-rhoai-connector**) to allow access to OCP resources like **routes**, **services**, and **inferenceservices**. For example:

     ```
     # Example for `ClusterRole`
     apiVersion: rbac.authorization.k8s.io/v1
     kind: ClusterRole
     metadata:
       name: rhdh-rhoai-connector
       annotations:
         argocd.argoproj.io/sync-wave: "0"
     rules:
     ```

```
  - apiGroups:
      - apiextensions.k8s.io
    resources:
      - customresourcedefinitions
    verbs:
      - get
  - apiGroups:
      - route.openshift.io
    resources:
      - routes
    verbs:
      - get
      - list
      - watch
  - apiGroups: [""]
    resources:
      - serviceaccounts
      - services
    verbs:
      - get
      - list
      - watch

  - apiGroups: ["serving.kserve.io"]
    resources: ["inferenceservices"]
    verbs: ["get", "list", "watch"]
```

```
# Example for `ClusterRoleBinding`
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: rhdh-rhoai-connector
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: rhdh-rhoai-connector
subjects:
  - kind: ServiceAccount
    name: rhdh-rhoai-connector
    namespace: ai-rhdh
```

- **Role** and **RoleBinding** to allow ConfigMap updates within the RHDH namespace (**ai-rhdh**). For example:

```
# Example for `Role` and `Rolebinding` in the {product-very-short} namespace (`ai-rhdh`)
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: rhdh-rhoai-connector
  namespace: ai-rhdh
rules:
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["get", "list", "watch", "create", "update", "patch"]
---
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: rhdh-rhoai-connector
  namespace: ai-rhdh
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: rhdh-rhoai-connector
subjects:
  - kind: ServiceAccount
    name: rhdh-rhoai-connector
    namespace: ai-rhdh
```

- **RoleBinding** in the RHOAI namespace (**rhoai-model-registries**) to grant the RHDH **ServiceAccount** read permissions to the model registry data (binding to **registry-user-modelregistry-public**).

```
# Example for `RoleBinding` in the {rhoai-short} namespace (rhoai-model-registries)
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  # if using ODH then change rhoai to odh in the name and namespace here
  name: rhdh-rhoai-dashboard-permissions
  # namespace: odh-model-registries
  namespace: rhoai-model-registries
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: registry-user-modelregistry-public
subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: Group
    name: system:serviceaccounts:ai-rhdh
```

- Secret (**rhdh-rhoai-connector-token**) of type **kubernetes.io/service-account-token** that goes along with the **rhdh-rhoai-connector ServiceAccount**.

```
apiVersion: v1
kind: Secret
metadata:
  name: rhdh-rhoai-connector-token
  namespace: ai-rhdh
  annotations:
    kubernetes.io/service-account.name: rhdh-rhoai-connector
type: kubernetes.io/service-account-token
```

2. Update your RHDH dynamic plugin configuration. The RHDH Pod requires two dynamic plugins.

   a. In your RHDH dynamic plugins ConfigMap, add the following code:

```
plugins:
  - disabled: false
    package: oci://ghcr.io/redhat-developer/rhdh-plugin-export-overlays/red-hat-developer-
hub-backstage-plugin-catalog-backend-module-model-catalog:bs_1.42.5__0.7.0!red-hat-
```

```
developer-hub-backstage-plugin-catalog-backend-module-model-catalog
  - disabled: false
    package: oci://ghcr.io/redhat-developer/rhdh-plugin-export-overlays/red-hat-developer-
hub-backstage-plugin-catalog-techdoc-url-reader-backend:bs_1.42.5__0.3.0!red-hat-
developer-hub-backstage-plugin-catalog-techdoc-url-reader-backend
```

3. Add the **Connector** sidecar containers to the RHDH Pod.
   - If RHDH was installed using the Operator, modify your RHDH custom resource (CR) instance.
   - If RHDH was installed using the Helm charts, modify the **Deployment** specification.
4. The system relies on three sidecar containers (OpenShift AI Connector for Red Hat Developer Hub) running alongside the **backstage-backend** container.

Add these sidecar containers to your configuration referencing the **rhdh-rhoai-connector-token** Secret: **location: Provides the REST API for RHDH plugins to fetch model metadatastorage-rest**: Maintains a cache of AI Model metadata in a ConfigMap called **bac-import-model**. ** **rhoai-normalizer**: Acts as a Kubernetes controller and RHOAI client, normalizing RHOAI metadata for the connector. The following code block is an example:

+

```
spec:
  template:
    spec:
      containers:
        - env:
            - name: NORMALIZER_FORMAT
              value: JsonArrayFormat
            - name: POD_IP
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
            - name: POD_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
          envFrom:
            - secretRef:
                name: rhdh-rhoai-connector-token
          image: quay.io/redhat-ai-dev/model-catalog-location-
service@sha256:c4471e07be6e0dbe821613053e6264a552cacda7f8604dbf306e6ac9e81e8ab9
          imagePullPolicy: Always
          name: location
          ports:
            - containerPort: 9090
              name: location
              protocol: TCP
          volumeMounts:
            - mountPath: /opt/app-root/src/dynamic-plugins-root
              name: dynamic-plugins-root
          workingDir: /opt/app-root/src
        - env:
            - name: NORMALIZER_FORMAT
          value: JsonArrayFormat
        - name: STORAGE_TYPE
          value: ConfigMap
        - name: BRIDGE_URL
          value: http://localhost:9090
        - name: POD_IP
          valueFrom:
            fieldRef:
              fieldPath: status.podIP
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
      envFrom:
        - secretRef:
            name: rhdh-rhoai-connector-token
      image: quay.io/redhat-ai-dev/model-catalog-storage-
rest@sha256:398095e7469e86d84b1196371286363f4b7668aa3e26370b4d78cb8d4ace1dc9
      imagePullPolicy: Always
      name: storage-rest
      volumeMounts:
        - mountPath: /opt/app-root/src/dynamic-plugins-root
          name: dynamic-plugins-root
      workingDir: /opt/app-root/src
    - env:
        - name: NORMALIZER_FORMAT
          value: JsonArrayFormat
        - name: POD_IP
          valueFrom:
            fieldRef:
              fieldPath: status.podIP
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
      envFrom:
        - secretRef:
            name: rhdh-rhoai-connector-token
      image: quay.io/redhat-ai-dev/model-catalog-rhoai-
normalizer@sha256:9f19742450a3a9c6d9c01d8341a20db7eb5a52a39348f488ae06b6aa49754a26
      imagePullPolicy: Always
      name: rhoai-normalizer
      volumeMounts:
        - mountPath: /opt/app-root/src/dynamic-plugins-root
          name: dynamic-plugins-root
      workingDir: /opt/app-root/src
      args:
        - '--metrics-address=:8081'
```

1. Enable **Connector** in your **RHDHapp-config.yaml** file. In your **Backstage `app-config.extra.yaml** file, configure **Entity Provider** under the **catalog.providers** section:

```
providers:
  modelCatalog:
    development:
      baseUrl: http://localhost:9090
```

where:

modelCatalog

Specifies the name of the provider.

development

Defines future connector capability beyond a single **baseUrl**.

baseUrl

For Developer Preview, this value is the only one supported. Future releases might support external routes.

CHAPTER 3. ENRICH AI MODEL METADATA FOR ENHANCED RED HAT DEVELOPER HUB EXPERIENCE

While RHOAI provides essential data, an AI platform engineer using RHOAI can enrich the Backstage/RHDH experience by adding **custom properties** to the **ModelVersion** or **RegisteredModel** (or annotations to the **KServe InferenceService** if the model registry is not used) so that the OpenShift AI Connector for Red Hat Developer Hub can add the information to the RHDH entities it creates. For more details, see Editing model version metadata in a model registry .

Property Key	Entity Field Populated	Description
API Spec	API Definition Tab	The OpenAPI / Swagger JSON specification for the model REST API.
API Type	API Type	Correlates to supported RHDH/Backstage API types (defaults to **openapi**).
TechDocs	TechDocs	URL pointing to a Git repository that follows RHDH TechDocs conventions for the Model Card. Use this setting only if the **Model Card to TechDocs** mapping is not active.
Homepage URL	Links	A URL considered the home page for the model.
Owner	Owner	Overrides the default OpenShift user as the entity owner.
Lifecycle	Lifecycle	Serves a means to express the lifecycle notion of RHDH/Backstage.
How to use	Links	A URL that points to usage documentation.
License	Links	A URL to the license file of the model.

Property Key	Entity Field Populated	Description
rhdh.modelcatalog.io/model-name	Annotations	A key piece of metadata is the name of the model used when communicating with the model server’s REST API. OpenShift AI Connector for Red Hat Developer Hub stores this name in the **rhdh.modelcatalog.io/model-name** annotation on the **Resource** entity, and will default this annotation’s value to the combined names of the **ResourceModel** and **ModelVersion** in the model registry, or the **KServe InferenceService** name if the model registry is not used, as those often line up with the model name used when communicating with the model server’s REST API. But the names are not guaranteed to match. When the names do not match, provide the correct model name for model REST API invocations.

3.1. POPULATING THE API DEFINITION TAB IN RHDH API ENTITIES

Because RHOAI does not expose the OpenAPI specification by default, the AI platform engineer can take the following steps to provide this information:

Procedure

1. Retrieve OpenAPI JSON: Use a tool such as **curl** to fetch the specification directly from the running endpoint of the AI model server. The following command provides the precise endpoint (/**openapi.json**) and shows how to include a **Bearer** token if the model requires authentication for access.

```
$ curl -k -H "Authorization: Bearer $MODEL_API_KEY"
https://$MODEL_ROOT_URL_INCLUDING_PORT/openapi.json | jq > open-api.json
```

2. Set Property in RHOAI.

   a. In the **RHOAI** dashboard, go to **Model Registry** and select the appropriate **Model Version**.

   NOTE

   We recommend using **Model Version** instead of **Registered Model** to maintain stability if the API changes between versions.

   b. In the **Properties** section, set a key/value pair where the key is **API Spec** and the value is the entire JSON content from the **open-api.json** file.

3. Propagation: The OpenShift AI Connector for Red Hat Developer Hub periodically polls the RHOAI Model Registry, propagates this JSON, and renders the interactive API documentation in the **Definition** tab of the RHDH API entity.

CHAPTER 4. TROUBLESHOOTING CONNECTOR FUNCTIONALITY

The connector system consists of the two dynamic plugins and the three OpenShift AI Connector for RHDH sidecar containers. Generally speaking, the logs collected must be provided to Red Hat Support for analysis.

The actual contents of the diagnostic data are not part of any product guaranteed specification, and can change at any time.

NOTE

During startup, you may see non-critical log errors, such as **in cluster config error: open /var/run/secrets/kubernetes.io/serviceaccount/token: no such file or directory**, in the sidecar logs. This error is expected during the initial setup and do not indicate a failure, provided the container eventually becomes healthy.

4.1. CHECKING DYNAMIC PLUGINS STATUS

Validate that the dynamic plugins have been successfully installed into your RHDH project Pod by using the following command:

```
$ oc logs -c install-dynamic-plugins deployment/<your RHDH deployment>
```

In the **install-dynamic-plugin** logs, you can check the following installation logs for successful logs:

- **red-hat-developer-hub-backstage-plugin-catalog-backend-module-model-catalog** (Entity Provider)
- **red-hat-developer-hub-backstage-plugin-catalog-techdoc-url-reader-backend** (TechDoc URL Reader)

4.2. INSPECTING PLUGIN LOGS

View the OpenShift AI Connector for Red Hat Developer Hub plugins in the **backstage-backend** container. Items to look for:

Plugin Component	Logger Service Target	Common Log Text
Model Catalog Entity Provider	**ModelCatalogResourceEntityProvider**	**Discovering ResourceEntities from Model Server...**
Model Catalog TechDoc URL Reader	**ModelCatalogBridgeTechdocUrlReader**	**ModelCatalogBridgeTechdocUrlReader.readUrl**

To enable debug logging, set the **LOG_LEVEL** environment variable to **debug** on the **backstage-backend** container. For more information, see Monitoring and logging.

4.3. INSPECTING THE OPENSHIFT AI CONNECTOR FOR RHDH

The OpenShift AI Connector for RHDH sidecars manage the data fetching and storage:

1. Check Cached Data (ConfigMap): The processed AI Model metadata is stored in a **ConfigMap**.

   ```
   $ oc get configmap bac-import-model -o json | jq -r '.binaryData | to_entries[] | "=== \(.key)
   ===\n" + (.value | @base64d | fromjson | .body | @base64d | fromjson | tostring)' | jq -R 'if
   startswith("=== ") then . else (. | fromjson) end'
   ```

2. Check Location Service API: Confirm the location service is providing data to the RHDH Entity Provider.

   ```
   $ oc rsh -c backstage-backend deployment/<your RHDH deployment>
   $ curl http://localhost:9090/list
   ```

3. Check Sidecar Container Logs:

   ```
   $ oc logs -c rhoai-normalizer deployment/<your {product-very-short} deployment>
   $ oc logs -c storage-rest deployment/<your {product-very-short} deployment>
   $ oc logs -c location deployment/<your {product-very-short} deployment>
   ```

4.4. OPENSHIFT AI MODEL REGISTRY AND MODEL CATALOG QUERIES

To access the same RHOAI data as the connector, use **curl** to query the RHOAI model registry and model catalog APIs, ensuring the **ServiceAccount** token has correct access control:

- Example showing how to fetch registered models

  ```
  $ curl -k -H "Authorization: Bearer $TOKEN"
  $RHOAI_MODEL_REGISTRY_URL/api/model_registry/v1alpha3/registered_models | jq
  ```

- Example showing how to fetch model versions

  ```
  $ curl -k -H "Authorization: Bearer $TOKEN"
  $RHOAI_MODEL_REGISTRY_URL/api/model_registry/v1alpha3/model_versions | jq
  ```

- Example showing how to fetch model artifacts

  ```
  $ curl -k -H "Authorization: Bearer $TOKEN"
  $RHOAI_MODEL_REGISTRY_URL/api/model_registry/v1alpha3/model_artifacts | jq
  ```

- Example showing how to fetch inference services

  ```
  $ curl -k -H "Authorization: Bearer $TOKEN"
  $RHOAI_MODEL_REGISTRY_URL/api/model_registry/v1alpha3/inference_services | jq
  ```

- Example showing how to fetch serving environments

  ```
  $ curl -k -H "Authorization: Bearer $TOKEN"
  $RHOAI_MODEL_REGISTRY_URL/api/model_registry/v1alpha3/serving_environments | jq
  ```

- Example showing how to fetch catalog sources

```
$ curl -k -H "Authorization: Bearer $TOKEN"
$RHOAI_MODEL_CATALOG_URL/api/model_catalog/v1alpha1/sources | jq
```