RED HAT
OPENSHIFT AI
SELF-MANAGED
2.7

SERVING MODELS

SERVE MODELS IN RED HAT OPENSHIFT AI SELF-MANAGED

Abstract

Serve models in Red Hat OpenShift AI Self-Managed. Serving trained models enables you to test and implement them into intelligent applications.

CHAPTER 1. ABOUT MODEL SERVING

Serving trained models on Red Hat OpenShift AI means deploying the models on your OpenShift cluster to test and then integrate them into intelligent applications. Deploying a model makes it available as a service that you can access by using an API. This enables you to return predictions based on data inputs that you provide through API calls. This process is known as model *inferencing*. When you serve a model on OpenShift AI, the inference endpoints that you can access for the deployed model are shown in the dashboard.

OpenShift AI provides the following model serving platforms:

Single model serving platform

For deploying large models such as large language models (LLMs), OpenShift AI includes a *single model serving platform* that is based on the KServe component. Because each model is deployed from its own model server, the single model serving platform helps you to deploy, monitor, scale, and maintain large models that require increased resources.

Multi-model serving platform

For deploying small and medium-sized models, OpenShift AI includes a *multi-model serving platform* that is based on the ModelMesh component. On the multi-model serving platform, you can deploy multiple models on the same model server. Each of the deployed models shares the server resources. This approach can be advantageous on OpenShift clusters that have finite compute resources or pods.

CHAPTER 2. SERVING SMALL AND MEDIUM-SIZED MODELS

For deploying small and medium-sized models, OpenShift AI includes a *multi-model serving platform* that is based on the ModelMesh component. On the multi-model serving platform, multiple models can be deployed from the same model server and share the server resources.

2.1. CONFIGURING MODEL SERVERS

2.1.1. Enabling the multi-model serving platform

To use the multi-model serving platform, you must first enable the platform.

Prerequisites

- You have logged in to Red Hat OpenShift AI.
- If you are using specialized OpenShift AI groups, you are part of the admin group (for example, `rhoai-admins`) in OpenShift.

Procedure

1. In the left menu of the OpenShift AI dashboard, click **Settings → Cluster settings**.
2. Locate the **Model serving platforms** section.
3. Select the **Multi-model serving platform** checkbox.
4. Click **Save changes**.

2.1.2. Adding a custom model-serving runtime for the multi-model serving platform

A model-serving runtime adds support for a specified set of model frameworks (that is, formats). By default, the multi-model serving platform includes the OpenVINO Model Server runtime. However, if this runtime doesn't meet your needs (it doesn't support a particular model format, for example), you can add your own, custom runtime.

As an administrator, you can use the Red Hat OpenShift AI dashboard to add and enable a custom model-serving runtime. You can then choose the custom runtime when you create a new model server for the multi-model serving platform.

> **Note**
>
> OpenShift AI enables you to add your own custom runtimes, but does not support the runtimes themselves. You are responsible for correctly configuring and maintaining custom runtimes. You are also responsible for ensuring that you are licensed to use any custom runtimes that you add.

Prerequisites

- You have logged in to OpenShift AI as an administrator.
- You are familiar with how to add a model server to your project. When you have added a custom model-serving runtime, you must configure a new model server to use the runtime.
- You have reviewed the example runtimes in the kserve/modelmesh-serving repository. You can use these examples as starting points. However, each runtime requires some further modification before you can deploy it in OpenShift AI. The required modifications are described in the following procedure.

  > **Note**

OpenShift AI includes the OpenVINO Model Server runtime by default. You do not need to add this runtime to OpenShift AI.

Procedure

1. From the OpenShift AI dashboard, click **Settings** > **Serving runtimes**.

   The **Serving runtimes** page opens and shows the model-serving runtimes that are already installed and enabled.

2. To add a custom runtime, choose one of the following options:

   - To start with an existing runtime (for example the OpenVINO Model Server runtime), click the action menu (⋮) next to the existing runtime and then click **Duplicate**.
   - To add a new custom runtime, click **Add serving runtime**.

3. In the **Select the model serving platforms this runtime supports** list, perform one of the following actions:

   - To add a custom runtime for only the multi-model serving platform, select **Multi-model serving platform**.
   - To add a custom runtime for both the multi- and single model serving platforms, select **Single-model and multi-model serving platforms**.

4. Optional: If you started a new runtime (rather than duplicating an existing one), add your code by choosing one of the following options:

   - **Upload a YAML file**

     - Click **Upload files**.
     - In the file browser, select a YAML file on your computer. This file might be the one of the example runtimes that you downloaded from the kserve/modelmesh-serving repository.

The embedded YAML editor opens and shows the contents of the file that you uploaded.

- **Enter YAML code directly in the editor**

  - Click **Start from scratch**.
  - Enter or paste YAML code directly in the embedded editor. The YAML that you paste might be copied from one of the example runtimes in the kserve/modelmesh-serving repository.

5. Optional: If you are adding one of the example runtimes in the kserve/modelmesh-serving repository, perform the following modifications:

   a. In the YAML editor, locate the **`kind`** field for your runtime. Update the value of this field to **`ServingRuntime`**.

   b. In the kustomization.yaml file in the kserve/modelmesh-serving repository, take note of the **`newName`** and **`newTag`** values for the runtime that you want to add. You will specify these values in a later step.

   c. In the YAML editor for your custom runtime, locate the **`containers.image`** field.

   d. Update the value of the **`containers.image`** field in the format **`newName:newTag`**, based on the values that you previously noted in the kustomization.yaml file. Some examples are shown.

   Nvidia Triton Inference Server

   ```
   image: nvcr.io/nvidia/tritonserver:23.04-py3
   ```

   Seldon Python MLServer

   ```
   image: seldonio/mlserver:1.3.2
   ```

   TorchServe

```
image: pytorch/torchserve:0.7.1-cpu
```

6. In the **`metadata.name`** field, ensure that the value of the runtime you are adding is unique (that is, the value doesn't match a runtime that you have already added).

7. Optional: To configure a custom display name for the runtime that you are adding, add a **`metadata.annotations.openshift.io/display-name`** field and specify a value, as shown in the following example:

```
apiVersion: serving.kserve.io/v1alpha1
kind: ServingRuntime
metadata:
  name: mlserver-0.x
  annotations:
    openshift.io/display-name: MLServer
```

Note

If you do not configure a custom display name for your runtime, OpenShift AI shows the value of the **`metadata.name`** field.

8. Click **Add**.

   The **Serving runtimes** page opens and shows the updated list of runtimes that are installed. Observe that the runtime you added is automatically enabled.

9. Optional: To edit your custom runtime, click the action menu (⋮) and select **Edit**.

Verification

- The model-serving runtime that you added is shown in an enabled state on the **Serving runtimes** page.

Additional resources

- To learn how to configure a model server that uses a custom model-serving runtime that you have added, see Adding a model server to your data science project.

2.1.3. Adding a model server for the multi-model serving platform

When you have enabled the multi-model serving platform, you must configure a model server to deploy models. If you require extra computing power for use with large datasets, you can assign accelerators to your model server.

Prerequisites

- You have logged in to Red Hat OpenShift AI.
- If you use specialized OpenShift AI groups, you are part of the user group or admin group (for example, **`rhoai-users`** or **`rhoai-admins`**) in OpenShift.
- You have created a data science project that you can add a model server to.
- You have enabled the multi-model serving platform.
- If you want to use a custom model-serving runtime for your model server, you have added and enabled the runtime. See Adding a custom model-serving runtime.
- If you want to use graphics processing units (GPUs) with your model server, you have enabled GPU support in OpenShift AI. See Enabling GPU support in OpenShift AI.

Procedure

1. In the left menu of the OpenShift AI dashboard, click **Data Science Projects**.
2. Click the name of the project that you want to configure a model server for.

   A project details page opens.
3. In the **Models and model servers** section, perform one of the following actions:
   - If you see a **Multi-model serving platform** tile, click **Add model server** on the tile.
   - If you do not see any tiles, click the **Add model server** button.

   The **Add model server** dialog opens.
4. In the **Model server name** field, enter a unique name for the model server.
5. From the **Serving runtime** list, select a model-serving runtime that is installed and enabled in your OpenShift AI deployment.

   Note

   If you are using a *custom* model-serving runtime with your model server and want to use GPUs, you must ensure that your custom runtime supports GPUs and is appropriately configured to use them.
6. In the **Number of model replicas to deploy** field, specify a value.
7. From the **Model server size** list, select a value.
8. Optional: If you selected **Custom** in the preceding step, configure the following settings in the **Model server size** section to customize your model server:

a. In the **CPUs requested** field, specify the number of CPUs to use with your model server. Use the list beside this field to specify the value in cores or millicores.

b. In the **CPU limit** field, specify the maximum number of CPUs to use with your model server. Use the list beside this field to specify the value in cores or millicores.

c. In the **Memory requested** field, specify the requested memory for the model server in gibibytes (Gi).

d. In the **Memory limit** field, specify the maximum memory limit for the model server in gibibytes (Gi).

9. Optional: From the **Accelerator** list, select an accelerator.

    a. If you selected an accelerator in the preceding step, specify the number of accelerators to use.

10. Optional: In the **Model route** section, select the **Make deployed models available through an external route** checkbox to make your deployed models available to external clients.

11. Optional: In the **Token authorization** section, select the **Require token authentication** checkbox to require token authentication for your model server. To finish configuring token authentication, perform the following actions:

    a. In the **Service account name** field, enter a service account name for which the token will be generated. The generated token is created and displayed in the **Token secret** field when the model server is configured.

    b. To add an additional service account, click **Add a service account** and enter another service account name.

12. Click **Add**.

    - The model server that you configured appears in the **Models and model servers** section of the project details page.

13. Optional: To update the model server, click the action menu (⋮) beside the model server and select **Edit model server**.

2.1.4. Deleting a model server

When you no longer need a model server to host models, you can remove it from your data science project.

> **Note**
>
> When you remove a model server, you also remove the models that are hosted on that model server. As a result, the models are no longer available to applications.

Prerequisites

- You have created a data science project and an associated model server.
- You have notified the users of the applications that access the models that the models will no longer be available.
- If you are using specialized OpenShift AI groups, you are part of the user group or admin group (for example, **`rhoai-users`** or **`rhoai-admins`**) in OpenShift.

Procedure

1. From the OpenShift AI dashboard, click **Data Science Projects**.

   The **Data science projects** page opens.

2. Click the name of the project from which you want to delete the model server.

   A project details page opens.

3. Click the action menu (⋮) beside the project whose model server you want to delete in the **Models and model servers** section and then click **Delete model server**.

The **Delete model server** dialog opens.

4. Enter the name of the model server in the text field to confirm that you intend to delete it.

5. Click **Delete model server**.

Verification

- The model server that you deleted is no longer displayed in the **Models and model servers** section on the project details page.

2.2. WORKING WITH DEPLOYED MODELS

2.2.1. Deploying a model by using the multi-model serving platform

You can deploy trained models on OpenShift AI to enable you to test and implement them into intelligent applications. Deploying a model makes it available as a service that you can access by using an API. This enables you to return predictions based on data inputs.

When you have enabled the multi-model serving platform, you can deploy models on the platform.

Prerequisites

- You have logged in to Red Hat OpenShift AI.
- If you are using specialized OpenShift AI groups, you are part of the user group or admin group (for example, **`rhoai-users`**) in OpenShift.
- You have enabled the multi-model serving platform.

- You have created a data science project and added a model server.
- You have access to S3-compatible object storage.
- For the model that you want to deploy, you know the associated folder path in your S3-compatible object storage bucket.

Procedure

1. In the left menu of the OpenShift AI dashboard, click **Data Science Projects**.
2. Click the name of the project that you want to deploy a model in.

   A project details page opens.
3. In the **Models and model servers** section, click **Deploy model**.
4. Configure properties for deploying your model as follows:

   a. In the **Model name** field, enter a unique name for the model that you are deploying.

   b. From the **Model framework** list, select a framework for your model.

   > **Note**
   >
   > The **Model framework** list shows only the frameworks that are supported by the model-serving runtime that you specified when you configured your model server.

   c. To specify the location of the model you want to deploy from S3-compatible object storage, perform one of the following sets of actions:

- **To use an existing data connection**
    - Select **Existing data connection**.
    - From the **Name** list, select a data connection that you previously defined.
    - In the **Path** field, enter the folder path that contains the model in your specified data source.
- **To use a new data connection**
    - To define a new data connection that your model can access, select **New data connection**.
    - In the **Name** field, enter a unique name for the data connection.
    - In the **Access key** field, enter the access key ID for the S3-compatible object storage provider.
    - In the **Secret key** field, enter the secret access key for the S3-compatible object storage account that you specified.
    - In the **Endpoint** field, enter the endpoint of your S3-compatible object storage bucket.
    - In the **Region** field, enter the default region of your S3-compatible object storage account.
    - In the **Bucket** field, enter the name of your S3-compatible object storage bucket.
    - In the **Path** field, enter the folder path in your S3-compatible object storage that contains your data file.

d. Click **Deploy**.

Verification

- Confirm that the deployed model is shown in the **Models and model servers** section of your project, and on the **Model Serving** page of the dashboard with a checkmark in the **Status** column.

2.2.2. Viewing a deployed model

To analyze the results of your work, you can view a list of deployed models on Red Hat OpenShift AI. You can also view the current statuses of deployed models and their endpoints.

Prerequisites

- You have logged in to Red Hat OpenShift AI.
- If you are using specialized OpenShift AI groups, you are part of the user group or admin group (for example, **`rhoai-users`** or **`rhoai-admins`**) in OpenShift.

Procedure

1. From the OpenShift AI dashboard, click **Model Serving**.

   The **Deployed models** page opens.

   For each model, the page shows details such as the model name, the project in which the model is deployed, the serving runtime that the model uses, and the deployment status.

2. Optional: For a given model, click the link in the **Inference endpoint** column to see the inference endpoints for the deployed model.

Verification

- A list of previously deployed data science models is displayed on the **Deployed models** page.

2.2.3. Updating the deployment properties of a deployed model

You can update the deployment properties of a model that has been deployed previously. This allows you to change the model's data connection and name.

Prerequisites

- You have logged in to Red Hat OpenShift AI.
- If you are using specialized OpenShift AI groups, you are part of the user group or admin group (for example, **`rhoai-users`** or **`rhoai-admins`**) in OpenShift.
- You have deployed a model on OpenShift AI.

Procedure

1. From the OpenShift AI dashboard, click **Model serving**.

   The **Model Serving** page opens.

2. Click the action menu (⋮) beside the model whose deployment properties you want to update and click **Edit**.

   The **Deploy model** dialog opens.

3. Update the deployment properties of the model as follows:

   a. In the **Model Name** field, enter a new, unique name for the model.

   b. From the **Model framework** list, select a framework for your model.

      Note

The **Model framework** list shows only the frameworks that are supported by the model-serving runtime that you specified when you configured your model server.

c. To update how you have specified the location of your model, perform one of the following sets of actions:

- **If you previously specified an existing data connection**
  - In the **Path** field, update the folder path that contains the model in your specified data source.
- **If you previously specified a new data connection**
  - In the **Name** field, update a unique name for the data connection.
  - In the **Access key** field, update the access key ID for the S3-compatible object storage provider.
  - In the **Secret key** field, update the secret access key for the S3-compatible object storage account that you specified.
  - In the **Endpoint** field, update the endpoint of your S3-compatible object storage bucket.
  - In the **Region** field, update the default region of your S3-compatible object storage account.
  - In the **Bucket** field, update the name of your S3-compatible object storage bucket.
  - In the **Path** field, update the folder path in your S3-compatible object storage that contains your data file.

d. Click **Deploy**.

Verification

- The model whose deployment properties you updated is displayed on the **Model Serving** page.

2.2.4. Deleting a deployed model

You can delete models you have previously deployed. This enables you to remove deployed models that are no longer required.

Prerequisites

- You have logged in to Red Hat OpenShift AI.
- If you are using specialized OpenShift AI groups, you are part of the user group or admin group (for example, **`rhoai-users`** or **`rhoai-admins`**) in OpenShift.
- You have deployed a model.

Procedure

1. From the OpenShift AI dashboard, click **Model serving**.

   The **Deployed models** page opens.

2. Click the action menu (⋮) beside the deployed model that you want to delete and click **Delete**.

   The **Delete deployed model** dialog opens.

3. Enter the name of the deployed model in the text field to confirm that you intend to delete it.
4. Click **Delete deployed model**.

Verification

- The model that you deleted is no longer displayed on the **Deployed models** page.

2.3. CONFIGURING MONITORING FOR THE MULTI MODEL SERVING PLATFORM

The multi model serving platform includes metrics for the ModelMesh component. When you have configured monitoring, you can grant Prometheus access to scrape the available metrics.

Prerequisites

- You have cluster administrator privileges for your OpenShift Container Platform cluster.
- You have downloaded and installed the OpenShift command-line interface (CLI). See Installing the OpenShift CLI.
- You are familiar with creating a config map for monitoring a user-defined workflow. You will perform similar steps in this procedure.
- You are familiar with enabling monitoring for user-defined projects in OpenShift. You will perform similar steps in this procedure.
- You have assigned the **`monitoring-rules-view`** role to users that will monitor metrics.

Procedure

1. In a terminal window, if you are not already logged in to your OpenShift cluster as a cluster administrator, log in to the OpenShift CLI as shown in the following example:

```
$ oc login <openshift_cluster_url> -u <admin_username> -p <password>
```

2. Define a **`ConfigMap`** object in a YAML file called **`uwm-cm-conf.yaml`** with the following contents:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      logLevel: debug
      retention: 15d
```

The **user-workload-monitoring-config** object configures the components that monitor user-defined projects. Observe that the retention time is set to the recommended value of 15 days.

3. Apply the configuration to create the **user-workload-monitoring-config** object.

```
$ oc apply -f uwm-cm-conf.yaml
```

4. Define another **ConfigMap** object in a YAML file called **uwm-cm-enable.yaml** with the following contents:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    enableUserWorkload: true
```

The **cluster-monitoring-config** object enables monitoring for user-defined projects.

5. Apply the configuration to create the **cluster-monitoring-config** object.

```
$ oc apply -f uwm-cm-enable.yaml
```

2.4. VIEWING METRICS FOR THE MULTI MODEL SERVING PLATFORM

When a cluster administrator has configured monitoring for the multi model serving platform, non-admin users can use the OpenShift web console to view metrics.

Prerequisites

- A cluster administrator has configured monitoring for the multi model serving platform.
- You have been assigned the **monitoring-rules-view** role.
- You are familiar with how to monitor project metrics in the OpenShift Container Platform web console.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Switch to the **Developer** perspective.
3. In the left menu, click **Observe**.
4. As described in monitoring project metrics, use the web console to run queries for **modelmesh_*** metrics.

CHAPTER 3. SERVING LARGE MODELS

For deploying large models such as large language models (LLMs), Red Hat OpenShift AI includes a *single model serving platform* that is based on the KServe component. Because each model is deployed from its own model server, the single model serving platform helps you to deploy, monitor, scale, and maintain large models that require increased resources.

3.1. ABOUT THE SINGLE MODEL SERVING PLATFORM

The single model serving platform consists of the following components:

- KServe: A Kubernetes custom resource definition (CRD) that orchestrates model serving for all types of models. It includes model-serving runtimes that implement the loading of given types of model servers. KServe handles the lifecycle of the deployment object, storage access, and networking setup.
- Red Hat OpenShift Serverless: A cloud-native development model that allows for serverless deployments of models. OpenShift Serverless is based on the open source Knative project.
- Red Hat OpenShift Service Mesh: A service mesh networking layer that manages traffic flows and enforces access policies. OpenShift Service Mesh is based on the open source Istio project.

To install the single model serving platform, you have the following options:

Automated installation

If you have not already created a **`ServiceMeshControlPlane`** or **`KNativeServing`** resource on your OpenShift cluster, you can configure the Red Hat OpenShift AI Operator to install KServe and its dependencies.

Manual installation

If you have already created a **`ServiceMeshControlPlane`** or **`KNativeServing`** resource on your OpenShift cluster, you *cannot* configure the Red Hat OpenShift AI Operator to install KServe and its dependencies. In this situation, you must install KServe manually.

When you have installed KServe, you can use the OpenShift AI dashboard to deploy models using pre-installed or custom model-serving runtimes.

OpenShift AI includes the following pre-installed runtimes for KServe:

- A standalone TGIS runtime
- A composite Caikit-TGIS runtime
- OpenVINO Model Server

Note

- Text Generation Inference Server (TGIS) is based on an early fork of Hugging Face TGI. Red Hat will continue to develop the standalone TGIS runtime to support TGI models. If a model does not work in the current version of OpenShift AI, support might be added in a future version. In the meantime, you can also add your own, custom runtime to support a TGI model. For more information, see Adding a custom model-serving runtime for the single model serving platform.

- The composite Caikit-TGIS runtime is based on Caikit and Text Generation Inference Server (TGIS). To use this runtime, you must convert your models to Caikit format. For an example, see Converting Hugging Face Hub models to Caikit format in the caikit-tgis-serving repository.

You can also configure monitoring for the single model serving platform and use Prometheus to scrape the available metrics.

3.2. CONFIGURING AUTOMATED INSTALLATION OF KSERVE

If you have not already created a **`ServiceMeshControlPlane`** or **`KNativeServing`** resource on your OpenShift cluster, you can configure the Red Hat OpenShift AI Operator to install KServe and its dependencies.

> **Important**
>
> If you have created a **`ServiceMeshControlPlane`** or **`KNativeServing`** resource on your cluster, the Red Hat OpenShift AI Operator cannot install KServe and its dependencies and the installation does not proceed. In this situation, you must follow the manual installation instructions to install KServe.

Prerequisites

- You have cluster administrator privileges for your OpenShift Container Platform cluster.
- Your cluster has a node with 4 CPUs and 16 GB memory.
- You have downloaded and installed the OpenShift command-line interface (CLI). For more information, see Installing the OpenShift CLI.

- You have installed the Red Hat OpenShift Service Mesh Operator and dependent Operators.

  Note

  To enable automated installation of KServe, install *only* the required Operators for Red Hat OpenShift Service Mesh. Do not perform any additional configuration or create a **`ServiceMeshControlPlane`** resource.

- You have installed the Red Hat OpenShift Serverless Operator.

  Note

  To enable automated installation of KServe, install *only* the Red Hat OpenShift Serverless Operator. Do not perform any additional configuration or create a **`KNativeServing`** resource.

- You have installed the Red Hat OpenShift AI Operator and created a **`DataScienceCluster`** object.

Procedure

1. Log in to the OpenShift web console as a cluster administrator.
2. In the web console, click **Operators → Installed Operators** and then click the Red Hat OpenShift AI Operator.
3. Install OpenShift Service Mesh as follows:
   a. Click the **DSC Initialization** tab.
   b. Click the **default-dsci** object.

c. Click the **YAML** tab.

d. In the **spec** section, validate that the value of the **managementState** field for the **serviceMesh** component is set to **Managed**, as shown:

```
spec:
 applicationsNamespace: redhat-ods-applications
 monitoring:
   managementState: Managed
   namespace: redhat-ods-monitoring
 serviceMesh:
   controlPlane:
     metricsCollection: Istio
     name: data-science-smcp
     namespace: istio-system
   managementState: Managed
```

Note

Do not change the **istio-system** namespace that is specified for the **serviceMesh** component by default. Other namespace values are not supported.

e. Click **Save**.

Based on the configuration you added to the **DSCInitialization** object, the Red Hat OpenShift AI Operator installs OpenShift Service Mesh.

4. Install both KServe and OpenShift Serverless as follows:

a. In the web console, click **Operators** → **Installed Operators** and then click the Red Hat OpenShift AI Operator.

b. Click the **Data Science Cluster** tab.

c. Click the **default-dsc** DSC object.

d. Click the **YAML** tab.

e. In the **`spec.components`** section, configure the **`kserve`** component as shown.

```
spec:
 components:
   kserve:
     managementState: Managed
     serving:
       ingressGateway:
         certificate:
           secretName: knative-serving-cert
           type: SelfSigned
       managementState: Managed
       name: knative-serving
```

f. Click **Save**.

The preceding configuration creates an ingress gateway for OpenShift Serverless to receive traffic from OpenShift Service Mesh. In this configuration, observe the following details:

- The configuration shown generates a self-signed certificate to secure incoming traffic to your OpenShift cluster and stores the certificate in the **`knative-serving-cert`** secret that is specified in the **`secretName`** field. To provide *your own* certificate, update the value of the **`secretName`** field to specify your secret name and change the value of the type field to **`Provided`**.

  Note

  If you provide your own certificate, the certificate must specify the domain name used by the ingress controller of your OpenShift cluster. You can check this value by running the following command:

  ```
  $ oc get ingresses.config.openshift.io cluster -o
  jsonpath='{.spec.domain}'
  ```

- You must set the value of the **`managementState`** field to **`Managed`** for both the **`kserve`** and **`serving`** components. Setting **`kserve.managementState`** to **`Managed`** triggers automated installation of KServe. Setting **`serving.managementState`** to **`Managed`** triggers automated installation of OpenShift Serverless. However, installation of OpenShift Serverless will *not* be triggered if **`kserve.managementState`** is not also set to **`Managed`**.

Verification

- Verify installation of OpenShift Service Mesh as follows:
  - In the web console, click **Workloads** → **Pods**.
  - From the project list, select **istio-system**. This is the project in which OpenShift Service Mesh is installed.
  - Confirm that there are running pods for the service mesh control plane, ingress gateway, and egress gateway. These pods have the naming patterns shown in the following example:

```
NAME                                                          READY
STATUS    RESTARTS   AGE
istio-egressgateway-7c46668687-fzsqj                          1/1
Running   0          22h
istio-ingressgateway-77f94d8f85-fhsp9                         1/1
Running   0          22h
istiod-data-science-smcp-cc8cfd9b8-2rkg4                      1/1
Running   0          22h
```

- Verify installation of OpenShift Serverless as follows:
  - In the web console, click **Workloads** → **Pods**.
  - From the project list, select **knative-serving**. This is the project in which OpenShift Serverless is installed.
  - Confirm that there are numerous running pods in the **`knative-serving`** project, including activator, autoscaler, controller, and domain mapping pods, as well as pods for the Knative Istio controller (which controls the integration of OpenShift Serverless and OpenShift Service Mesh). An example is shown.

```
NAME                                                READY     STATUS
RESTARTS  AGE
activator-7586f6f744-nvdlb                          2/2       Running   0
22h
activator-7586f6f744-sd77w                          2/2       Running   0
22h
autoscaler-764fdf5d45-p2v98                         2/2       Running   0
22h
autoscaler-764fdf5d45-x7dc6                         2/2       Running   0
22h
```

```
autoscaler-hpa-7c7c4cd96d-2lkzg                 1/1       Running   0
22h
autoscaler-hpa-7c7c4cd96d-gks9j                 1/1       Running   0
22h
controller-5fdfc9567c-6cj9d                     1/1       Running   0
22h
controller-5fdfc9567c-bf5x7                     1/1       Running   0
22h
domain-mapping-56ccd85968-2hjvp                 1/1       Running   0
22h
domain-mapping-56ccd85968-lg6mw                 1/1       Running   0
22h
domainmapping-webhook-769b88695c-gp2hk          1/1       Running   0
22h
domainmapping-webhook-769b88695c-npn8g          1/1       Running   0
22h
net-istio-controller-7dfc6f668c-jb4xk           1/1       Running   0
22h
net-istio-controller-7dfc6f668c-jxs5p           1/1       Running   0
22h
net-istio-webhook-66d8f75d6f-bgd5r              1/1       Running   0
22h
net-istio-webhook-66d8f75d6f-hld75              1/1       Running   0
22h
webhook-7d49878bc4-8xjbr                        1/1       Running   0
22h
webhook-7d49878bc4-s4xx4                        1/1       Running   0
22h
```

- Verify installation of KServe as follows:
  - In the web console, click **Workloads** → **Pods**.
  - From the project list, select **redhat-ods-applications**.This is the project in which OpenShift AI components are installed, including KServe.
  - Confirm that the project includes a running pod for the KServe controller manager, similar to the following example:

```
NAME                                          READY   STATUS
RESTARTS   AGE
kserve-controller-manager-7fbb7bccd4-t4c5g    1/1     Running   0
22h
odh-model-controller-6c4759cc9b-cftmk         1/1     Running   0
129m
odh-model-controller-6c4759cc9b-ngj8b         1/1     Running   0
129m
odh-model-controller-6c4759cc9b-vnhq5         1/1     Running   0
129m
```

3.3. MANUALLY INSTALLING KSERVE

If you have already installed the Red Hat OpenShift Service Mesh Operator and created a **`ServiceMeshControlPlane`** resource *or* if you have installed the Red Hat OpenShift Serverless Operator and created a **`KNativeServing`** resource, the Red Hat OpenShift AI Operator cannot install KServe and its dependencies. In this situation, you must install KServe manually.

Important

The procedures in this section show how to perform a *new* installation of KServe and its dependencies and are intended as a complete installation and configuration reference. If you have already installed and configured OpenShift Service Mesh or OpenShift Serverless, you might not need to follow all steps. If you are unsure about what updates to apply to your existing configuration to use KServe, contact Red Hat Support.

3.3.1. Installing KServe dependencies

Before you install KServe, you must install and configure some dependencies. Specifically, you must create Red Hat OpenShift Service Mesh and Knative Serving instances and then configure secure gateways for Knative Serving.

3.3.1.1. Creating an OpenShift Service Mesh instance

The following procedure shows how to create a Red Hat OpenShift Service Mesh instance.

Prerequisites

- You have cluster administrator privileges for your OpenShift Container Platform cluster.
- Your cluster has a node with 4 CPUs and 16 GB memory.
- You have downloaded and installed the OpenShift command-line interface (CLI). See Installing the OpenShift CLI.
- You have installed the Red Hat OpenShift Service Mesh Operator and dependent Operators.

Procedure

1. In a terminal window, if you are not already logged in to your OpenShift cluster as a cluster administrator, log in to the OpenShift CLI as shown in the following example:

```
$ oc login <openshift_cluster_url> -u <admin_username> -p <password>
```

2. Create the required namespace for Red Hat OpenShift Service Mesh.

```
$ oc create ns istio-system
```

You see the following output:

```
namespace/istio-system created
```

3. Define a **ServiceMeshControlPlane** object in a YAML file named **smcp.yaml** with the following contents:

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: minimal
  namespace: istio-system
spec:
  tracing:
    type: None
  addons:
    grafana:
      enabled: false
    kiali:
      name: kiali
```

```
    enabled: false
  prometheus:
    enabled: false
  jaeger:
    name: jaeger
security:
  dataPlane:
    mtls: true
  identity:
    type: ThirdParty
techPreview:
  meshConfig:
    defaultConfig:
      terminationDrainDuration: 35s
gateways:
  ingress:
    service:
      metadata:
        labels:
          knative: ingressgateway
proxy:
  networking:
    trafficControl:
      inbound:
        excludedPorts:
          - 8444
          - 8022
```

For more information about the values in the YAML file, see the Service Mesh control plane configuration reference.

4. Create the service mesh control plane.

```
$ oc apply -f smcp.yaml
```

Verification

- Verify creation of the service mesh instance as follows:
    - In the OpenShift CLI, enter the following command:

    ```
    $ oc get pods -n istio-system
    ```

    The preceding command lists all running pods in the **istio-system** project. This is the project in which OpenShift Service Mesh is installed.

    - Confirm that there are running pods for the service mesh control plane, ingress gateway, and egress gateway. These pods have the following naming patterns:

    ```
    NAME                                        READY   STATUS
    RESTARTS    AGE
    istio-egressgateway-7c46668687-fzsqj        1/1     Running    0
    22h
    istio-ingressgateway-77f94d8f85-fhsp9       1/1     Running    0
    22h
    istiod-data-science-smcp-cc8cfd9b8-2rkg4    1/1     Running    0
    22h
    ```

3.3.1.2. Creating a Knative Serving instance

The following procedure shows how to install Knative Serving and then create an instance.

Prerequisites

- You have cluster administrator privileges for your OpenShift Container Platform cluster.
- Your cluster has a node with 4 CPUs and 16 GB memory.
- You have downloaded and installed the OpenShift command-line interface (CLI). See Installing the OpenShift CLI.
- You have created a Red Hat OpenShift Service Mesh instance.
- You have installed the Red Hat OpenShift Serverless Operator.

Procedure

1. In a terminal window, if you are not already logged in to your OpenShift cluster as a cluster administrator, log in to the OpenShift CLI as shown in the following example:

```
$ oc login <openshift_cluster_url> -u <admin_username> -p <password>
```

2. Check whether the required project (that is, *namespace*) for Knative Serving already exists.

```
$ oc get ns knative-serving
```

If the project exists, you see output similar to the following example:

```
NAME              STATUS   AGE
knative-serving   Active   4d20h
```

3. If the **knative-serving** project *doesn't* already exist, create it.

```
$ oc create ns knative-serving
```

You see the following output:

```
namespace/knative-serving created
```

4. Define a **ServiceMeshMember** object in a YAML file called **default-smm.yaml** with the following contents:

```
apiVersion: maistra.io/v1
kind: ServiceMeshMember
metadata:
  name: default
  namespace: knative-serving
spec:
  controlPlaneRef:
    namespace: istio-system
    name: minimal
```

5. Create the **ServiceMeshMember** object in the **istio-system** namespace.

```
$ oc apply -f default-smm.yaml
```

You see the following output:

```
servicemeshmember.maistra.io/default created
```

6. Define a **KnativeServing** object in a YAML file called **knativeserving-istio.yaml** with the following contents:

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
  annotations:
    serverless.openshift.io/default-enable-http2: "true"
spec:
  workloads:
    - name: net-istio-controller
      env:
        - container: controller
          envVars:
            - name: ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID
              value: 'true'
    - annotations:
        sidecar.istio.io/inject: "true"  1
        sidecar.istio.io/rewriteAppHTTPProbers: "true  2
      name: activator
    - annotations:
        sidecar.istio.io/inject: "true"
        sidecar.istio.io/rewriteAppHTTPProbers: "true"
      name: autoscaler
  ingress:
    istio:
```

```
      enabled: true
  config:
    features:
      kubernetes.podspec-affinity: enabled
      kubernetes.podspec-nodeselector: enabled
      kubernetes.podspec-tolerations: enabled
```

The preceding file defines a custom resource (CR) for a **KnativeServing** object. The CR also adds the following actions to each of the activator and autoscaler pods:

1

Injects an Istio sidecar to the pod. This makes the pod part of the service mesh.

2

Enables the Istio sidecar to rewrite the HTTP liveness and readiness probes for the pod.

Note

If you configure a custom domain for a Knative service, you can use a TLS certificate to secure the mapped service. To do this, you must create a TLS secret, and then update the **DomainMapping** CR to use the TLS secret that you have created. For more information, see Securing a mapped service using a TLS certificate in the Red Hat OpenShift Serverless documentation.

7. Create the **KnativeServing** object in the specified **knative-serving** namespace.

```
$ oc apply -f knativeserving-istio.yaml
```

You see the following output:

```
knativeserving.operator.knative.dev/knative-serving created
```

Verification

- Review the default **ServiceMeshMemberRoll** object in the **istio-system** namespace.

  ```
  $ oc describe smmr default -n istio-system
  ```

  In the description of the **ServiceMeshMemberRoll** object, locate the **Status.Members** field and confirm that it includes the **knative-serving** namespace.

- Verify creation of the Knative Serving instance as follows:

  - In the OpenShift CLI, enter the following command:

    ```
    $ oc get pods -n knative-serving
    ```

    The preceding command lists all running pods in the **knative-serving** project. This is the project in which you created the Knative Serving instance.

  - Confirm that there are numerous running pods in the **knative-serving** project, including activator, autoscaler, controller, and domain mapping pods, as well as pods for the Knative Istio controller, which controls the integration of OpenShift Serverless and OpenShift Service Mesh. An example is shown.

    ```
    NAME                                                        READY       STATUS
    RESTARTS        AGE
    activator-7586f6f744-nvdlb                                  2/2         Running
    0               22h
    ```

```
activator-7586f6f744-sd77w                           2/2         Running
0                22h
autoscaler-764fdf5d45-p2v98                          2/2         Running
0                22h
autoscaler-764fdf5d45-x7dc6                          2/2         Running
0                22h
autoscaler-hpa-7c7c4cd96d-2lkzg                      1/1         Running
0                22h
autoscaler-hpa-7c7c4cd96d-gks9j                      1/1         Running
0                22h
controller-5fdfc9567c-6cj9d                          1/1         Running
0                22h
controller-5fdfc9567c-bf5x7                          1/1         Running
0                22h
domain-mapping-56ccd85968-2hjvp                      1/1         Running
0                22h
domain-mapping-56ccd85968-lg6mw                      1/1         Running
0                22h
domainmapping-webhook-769b88695c-gp2hk               1/1         Running
0                22h
domainmapping-webhook-769b88695c-npn8g               1/1         Running
0                22h
net-istio-controller-7dfc6f668c-jb4xk                1/1         Running
0                22h
net-istio-controller-7dfc6f668c-jxs5p                1/1         Running
0                22h
net-istio-webhook-66d8f75d6f-bgd5r                   1/1         Running
0                22h
```

```
net-istio-webhook-66d8f75d6f-hld75                1/1         Running
0                22h
webhook-7d49878bc4-8xjbr                          1/1         Running
0                22h
webhook-7d49878bc4-s4xx4                          1/1         Running
0                22h
```

3.3.1.3. Creating secure gateways for Knative Serving

To secure traffic between your Knative Serving instance and the service mesh, you must create secure gateways for your Knative Serving instance.

The following procedure shows how to use OpenSSL to generate a wildcard certificate and key and then use them to create local and ingress gateways for Knative Serving.

> **Important**
>
> If you have your own wildcard certificate and key to specify when configuring the gateways, you can skip to step 11 of this procedure.

Prerequisites

- You have cluster administrator privileges for your OpenShift Container Platform cluster.
- You have downloaded and installed the OpenShift command-line interface (CLI). See Installing the OpenShift CLI.
- You have created a Red Hat OpenShift Service Mesh instance.

- You have created a Knative Serving instance.
- If you intend to generate a wildcard certificate and key, you have downloaded and installed OpenSSL.

Procedure

1. In a terminal window, if you are not already logged in to your OpenShift cluster as a cluster administrator, log in to the OpenShift CLI as shown in the following example:

   ```
   $ oc login <openshift_cluster_url> -u <admin_username> -p <password>
   ```

   Important

   If you have your own wildcard certificate and key to specify when configuring the gateways, skip to step 11 of this procedure.

2. Set environment variables to define base directories for generation of a wildcard certificate and key for the gateways.

   ```
   $ export BASE_DIR=/tmp/kserve
   $ export BASE_CERT_DIR=${BASE_DIR}/certs
   ```

3. Set an environment variable to define the common name used by the ingress controller of your OpenShift cluster.

   ```
   $ export COMMON_NAME=$(oc get ingresses.config.openshift.io cluster -o
   jsonpath='{.spec.domain}' | awk -F'.' '{print $(NF-1)"."$NF}')
   ```

4. Set an environment variable to define the domain name used by the ingress controller of your OpenShift cluster.

```
$ export DOMAIN_NAME=$(oc get ingresses.config.openshift.io cluster -o
jsonpath='{.spec.domain}')
```

5. Create the required base directories for the certificate generation, based on the environment variables that you previously set.

```
$ mkdir ${BASE_DIR}
$ mkdir ${BASE_CERT_DIR}
```

6. Create the OpenSSL configuration for generation of a wildcard certificate.

```
$ cat <<EOF> ${BASE_DIR}/openssl-san.config
[ req ]
distinguished_name = req
[ san ]
subjectAltName = DNS:*.${DOMAIN_NAME}
EOF
```

7. Generate a root certificate.

```
$ openssl req -x509 -sha256 -nodes -days 3650 -newkey rsa:2048 \
-subj "/O=Example Inc./CN=${COMMON_NAME}" \
-keyout $BASE_DIR/root.key \
-out $BASE_DIR/root.crt
```

8. Generate a wildcard certificate signed by the root certificate.

```
$ openssl req -x509 -newkey rsa:2048 \
-sha256 -days 3560 -nodes \
-subj "/CN=${COMMON_NAME}/O=Example Inc." \
-extensions san -config ${BASE_DIR}/openssl-san.config \
-CA $BASE_DIR/root.crt \
-CAkey $BASE_DIR/root.key \
-keyout $BASE_DIR/wildcard.key  \
-out $BASE_DIR/wildcard.crt

$ openssl x509 -in ${BASE_DIR}/wildcard.crt -text
```

9. Verify the wildcard certificate.

```
$ openssl verify -CAfile ${BASE_DIR}/root.crt ${BASE_DIR}/wildcard.crt
```

10. Export the wildcard key and certificate that were created by the script to new environment variables.

```
$ export TARGET_CUSTOM_CERT=${BASE_CERT_DIR}/wildcard.crt
$ export TARGET_CUSTOM_KEY=${BASE_CERT_DIR}/wildcard.key
```

11. Optional: To export *your own* wildcard key and certificate to new environment variables, enter the following commands:

```
$ export TARGET_CUSTOM_CERT=<path_to_certificate>
$ export TARGET_CUSTOM_KEY=<path_to_key>
```

Note

In the certificate that you provide, you must specify the domain name used by the ingress controller of your OpenShift cluster. You can check this value by running the following command:

```
$ oc get ingresses.config.openshift.io cluster -o jsonpath='{.spec.domain}'
```

12. Create a TLS secret in the **istio-system** namespace using the environment variables that you set for the wildcard certificate and key.

```
$ oc create secret tls wildcard-certs --cert=${TARGET_CUSTOM_CERT} --
key=${TARGET_CUSTOM_KEY} -n istio-system
```

13. Create a **gateways.yaml** YAML file with the following contents:

```
apiVersion: v1
kind: Service  1
metadata:
  labels:
    experimental.istio.io/disable-gateway-port-translation: "true"
  name: knative-local-gateway
  namespace: istio-system
spec:
  ports:
    - name: http2
      port: 80
      protocol: TCP
      targetPort: 8081
  selector:
    knative: ingressgateway
  type: ClusterIP
```

```
---
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: knative-ingress-gateway  2
  namespace: knative-serving
spec:
  selector:
    knative: ingressgateway
  servers:
    - hosts:
        - '*'
      port:
        name: https
        number: 443
        protocol: HTTPS
      tls:
        credentialName: wildcard-certs
        mode: SIMPLE
---
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
 name: knative-local-gateway  3
 namespace: knative-serving
spec:
 selector:
   knative: ingressgateway
```

```
servers:
  - port:
      number: 8081
      name: https
      protocol: HTTPS
    tls:
      mode: ISTIO_MUTUAL
    hosts:
      - "*"
```

1

Defines a service in the **istio-system** namespace for the Knative local gateway.

2

Defines an ingress gateway in the **knative-serving namespace**. The gateway uses the TLS secret you created earlier in this procedure. The ingress gateway handles external traffic to Knative.

3

Defines a local gateway for Knative in the **knative-serving** namespace.

14. Apply the **gateways.yaml** file to create the defined resources.

```
$ oc apply -f gateways.yaml
```

You see the following output:

```
service/knative-local-gateway created
gateway.networking.istio.io/knative-ingress-gateway created
gateway.networking.istio.io/knative-local-gateway created
```

Verification

- Review the gateways that you created.

```
$ oc get gateway --all-namespaces
```

Confirm that you see the local and ingress gateways that you created in the **knative-serving** namespace, as shown in the following example:

```
NAMESPACE            NAME                            AGE
knative-serving      knative-ingress-gateway         69s
knative-serving    knative-local-gateway      2m
```

3.3.2. Installing KServe

To complete manual installation of KServe, you must install the Red Hat OpenShift AI Operator. Then, you can configure the Operator to install KServe.

Prerequisites

- You have cluster administrator privileges for your OpenShift Container Platform cluster.
- Your cluster has a node with 4 CPUs and 16 GB memory.
- You have downloaded and installed the OpenShift command-line interface (CLI). See Installing the OpenShift CLI.

- You have created a Red Hat OpenShift Service Mesh instance.
- You have created a Knative Serving instance.
- You have created secure gateways for Knative Serving.
- You have installed the Red Hat OpenShift AI Operator and created a **`DataScienceCluster`** object.

Procedure

1. Log in to the OpenShift web console as a cluster administrator.
2. In the web console, click **Operators → Installed Operators** and then click the Red Hat OpenShift AI Operator.
3. For installation of KServe, configure the OpenShift Service Mesh component as follows:
    a. Click the **DSC Initialization** tab.
    b. Click the **default-dsci** object.
    c. Click the **YAML** tab.
    d. In the **`spec`** section, add and configure the **`serviceMesh`** component as shown:

    ```
    spec:
     serviceMesh:
       managementState: Unmanaged
    ```

    e. Click **Save**.
4. For installation of KServe, configure the KServe and OpenShift Serverless components as follows:

a. In the web console, click **Operators** → **Installed Operators** and then click the Red Hat OpenShift AI Operator.

b. Click the **Data Science Cluster** tab.

c. Click the **default-dsc** DSC object.

d. Click the **YAML** tab.

e. In the **`spec.components`** section, configure the **`kserve`** component as shown:

```
spec:
 components:
   kserve:
     managementState: Managed
```

f. Within the **`kserve`** component, add the **`serving`** component, and configure it as shown:

```
spec:
 components:
   kserve:
     managementState: Managed
     serving:
       managementState: Unmanaged
```

g. Click **Save**.

3.4. DEPLOYING MODELS BY USING THE SINGLE MODEL SERVING PLATFORM

On the single model serving platform, each model is deployed from its own model server. This helps you to deploy, monitor, scale, and maintain LLMs that require increased resources.

> **Important**
>
> If you want to use a self-signed certificate to deploy a model from S3-compatible storage, the single model serving platform (which uses KServe) requires additional configuration. For more information, see the Red Hat Knowledgebase solution article How to use self-signed certificates with KServe.
>
> Alternatively, you can disable SSL authentication for KServe. For more information, see the Red Hat Knowledgebase solution article How to skip the validation of SSL for KServe.

3.4.1. Enabling the single model serving platform

When you have installed KServe, you can use the Red Hat OpenShift AI dashboard to enable the single model serving platform. You can also use the dashboard to enable model-serving runtimes for the platform.

Prerequisites

- You have logged in to Red Hat OpenShift AI.
- If you are using specialized OpenShift AI groups, you are part of the admin group (for example, `rhoai-admins`) in OpenShift.

- You have installed KServe.

Procedure

1. Enable the single model serving platform as follows:

    a. In the left menu, click **Settings → Cluster settings**.

    b. Locate the **Model serving platforms** section.

    c. To enable the single model serving platform for projects, select the **Single model serving platform** checkbox.

    d. Click **Save changes**.

2. Enable pre-installed runtimes for the single-model serving platform as follows:

    a. In the left menu of the OpenShift AI dashboard, click **Settings → Serving runtimes**.

    The **Serving runtimes** page shows any custom runtimes that you have added, as well as the following pre-installed runtimes:

    - **Caikit TGIS ServingRuntime for KServe**
    - **OpenVINO Model Server**
    - **TGIS Standalone ServingRuntime for KServe (gRPC)**

    b. Set the runtime that you want to use to **Enabled**.

    The single model serving platform is now available for model deployments.

3.4.2. Adding a custom model-serving runtime for the single model serving platform

A model-serving runtime adds support for a specified set of model frameworks (that is, formats). You have the option of using the pre-installed runtimes included with OpenShift AI or adding your own, custom runtimes. This is useful in instances where the pre-installed runtimes don't meet your needs. For example, you might find that the TGIS runtime does not support a particular model format that is supported by Hugging Face Text Generation Inference (TGI). In this case, you can create a custom runtime to add support for the model.

As an administrator, you can use the OpenShift AI interface to add and enable a custom model-serving runtime. You can then choose the custom runtime when you deploy a model on the single model serving platform.

> **Note**
>
> OpenShift AI enables you to add your own custom runtimes, but does not support the runtimes themselves. You are responsible for correctly configuring and maintaining custom runtimes. You are also responsible for ensuring that you are licensed to use any custom runtimes that you add.

Prerequisites

- You have logged in to OpenShift AI as an administrator.
- You have built your custom runtime and added the image to a container image repository such as Quay.

Procedure

1. From the OpenShift AI dashboard, click **Settings** > **Serving runtimes**.

   The **Serving runtimes** page opens and shows the model-serving runtimes that are already installed and enabled.

2. To add a custom runtime, choose one of the following options:

   - To start with an existing runtime (for example, **TGIS Standalone ServingRuntime for KServe**), click the action menu (⋮) next to the existing runtime and then click **Duplicate**.

   - To add a new custom runtime, click **Add serving runtime**.

3. In the **Select the model serving platforms this runtime supports** list, perform one of the following actions:

   - To add a runtime for only the single model serving platform, select **Single-model serving platform**.

   - To add a runtime for both the single- and multi-model serving platforms, select **Single-model and multi-model serving platforms**.

4. Optional: If you started a new runtime (rather than duplicating an existing one), add your code by choosing one of the following options:

   - **Upload a YAML file**

     - Click **Upload files**.

     - In the file browser, select a YAML file on your computer.

       The embedded YAML editor opens and shows the contents of the file that you uploaded.

   - **Enter YAML code directly in the editor**

     - Click **Start from scratch**.

     - Enter or paste YAML code directly in the embedded editor.

       Note

In many cases, creating a custom runtime will require adding new or custom parameters to the **env** section of the **ServingRuntime** specification.

5. Click **Add**.

   The **Serving runtimes** page opens and shows the updated list of runtimes that are installed. Observe that the runtime you added is automatically enabled.

6. Optional: To edit your custom runtime, click the action menu (⋮) and select **Edit**.

Verification

- The model-serving runtime that you added is shown in an enabled state on the **Serving runtimes** page.

3.4.3. Deploying models on the single model serving platform

When you have enabled the single model serving platform, you can enable a pre-installed or custom model-serving runtime and start to deploy models on the platform.

Note

Text Generation Inference Server (TGIS) is based on an early fork of Hugging Face TGI. Red Hat will continue to develop the standalone TGIS runtime to support TGI models. If a model does not work in the current version of OpenShift AI, support might be added in a future version. In the meantime, you can also add your own, custom runtime to support a TGI model. For more information, see Adding a custom model-serving runtime for the single model serving platform.

Prerequisites

- You have logged in to Red Hat OpenShift AI.
- If you are using specialized OpenShift AI groups, you are part of the user group or admin group (for example, **`rhoai-users`** or **`rhoai-admins`**) in OpenShift.
- You have installed KServe.
- You have enabled the single model serving platform.
- You have created a data science project.
- To use the Caikit-TGIS runtime, you have converted your model to Caikit format. For an example, see Converting Hugging Face Hub models to Caikit format in the caikit-tgis-serving repository.
- You know the folder path for the data connection that you want the model to access.
- If you want to use graphics processing units (GPUs) with your model server, you have enabled GPU support in OpenShift AI. See Enabling GPU support in OpenShift AI.

Procedure

1. In the left menu, click **Data Science Projects**.
2. Click the name of the project that you want to deploy a model in.
3. In the **Models and model servers** section, perform one of the following actions:
   - If you see a **Single model serving platform** tile, click **Deploy model** on the tile.
   - If you do not see any tiles, click the **Deploy model** button.

   The **Deploy model** dialog opens.
4. Configure properties for deploying your model as follows:

a. In the **Model name** field, enter a unique name for the model that you are deploying.

b. In the **Serving runtime** field, select an enabled runtime.

c. From the **Model framework** list, select a value.

d. In the **Number of model replicas to deploy** field, specify a value.

e. From the **Model server size** list, select a value.

f. To specify the location of your model, perform one of the following sets of actions:

- **To use an existing data connection**

    - Select **Existing data connection**.
    - From the **Name** list, select a data connection that you previously defined.
    - In the **Path** field, enter the folder path that contains the model in your specified data source.

- **To use a new data connection**

    - To define a new data connection that your model can access, select **New data connection**.
    - In the **Name** field, enter a unique name for the data connection.
    - In the **Access key** field, enter the access key ID for your S3-compatible object storage provider.
    - In the **Secret key** field, enter the secret access key for the S3-compatible object storage account that you specified.
    - In the **Endpoint** field, enter the endpoint of your S3-compatible object storage bucket.
    - In the **Region** field, enter the default region of your S3-compatible object storage account.

- In the **Bucket** field, enter the name of your S3-compatible object storage bucket.
- In the **Path** field, enter the folder path in your S3-compatible object storage that contains your data file.

g. Click **Deploy**.

Verification

- Confirm that the deployed model is shown in the **Models and model servers** section of your project, and on the **Model Serving** page of the dashboard with a check mark in the **Status** column.

3.4.4. Accessing the inference endpoints for models deployed on the single model serving platform

When you deploy a model by using the single model serving platform, the model is available as a service that you can access using API requests. This enables you to return predictions based on data inputs. To use API requests to interact with your deployed model, you must know how to access the inference endpoints that are available.

Prerequisites

- You have logged in to Red Hat OpenShift AI.
- If you are using specialized OpenShift AI groups, you are part of the user group or admin group (for example, **`rhoai-users`** or **`rhoai-admins`**) in OpenShift.
- You have deployed a model by using the single model serving platform.

Procedure

1. From the OpenShift AI dashboard, click **Model Serving**.

2. From the **Project** list, select the project that you deployed a model in.

3. In the **Deployed models** table, for the model that you want to access, copy the URL shown in the **Inference endpoint** column.

4. Depending on what action you want to perform with the model (and if the model supports that action), add one of the following paths to the end of the inference endpoint URL:

   Caikit TGIS ServingRuntime for KServe

   - **`:443/api/v1/task/text-generation`**
   - **`:443/api/v1/task/server-streaming-text-generation`**

   TGIS Standalone ServingRuntime for KServe (gRPC)

   - **`:443 fmaas.GenerationService/Generate`**
   - **`:443 fmaas.GenerationService/GenerateStream`**

   Note

   To query the endpoints for the TGIS standalone runtime, you must also download the files in the **`proto`** directory of the IBM text-generation-inference repository.

   OpenVINO Model Server

   - **`/v2/models/<model-name>/infer`**

As indicated by the paths shown, the single model serving platform uses the HTTPS port of your OpenShift router (usually port 443) to serve external API requests.

5. Use the endpoints to make API requests to your deployed model, as shown in the following example commands:

Caikit TGIS ServingRuntime for KServe

```
curl --json '{"model_id": "<model_name>", "inputs": "<text>"}'
https://<inference_endpoint_url>:443/api/v1/task/server-streaming-text-
generation
```

TGIS Standalone ServingRuntime for KServe (gRPC)

```
grpcurl -proto text-generation-inference/proto/generation.proto -d
'{"requests": [{"text":"<text>"}]}' -H 'mm-model-id: <model_name>' -
insecure <inference_endpoint_url>:443 fmaas.GenerationService/Generate
```

OpenVINO Model Server

```
curl -ks <inference_endpoint_url>/v2/models/<model_name>/infer -d '{
"model_name": "<model_name>", "inputs": [{ "name": "
<name_of_model_input>", "shape": [<shape>], "datatype": "<data_type>",
"data": [<data>] }]}'
```

Additional resources

- Text Generation Inference Server (TGIS)
- Caikit API documentation
- OpenVINO KServe-compatible REST API documentation

3.5. CONFIGURING MONITORING FOR THE SINGLE MODEL SERVING PLATFORM

The single model serving platform includes metrics for Caikit and TGIS. You can also configure monitoring for OpenShift Service Mesh. The service mesh metrics helps you to understand dependencies and traffic flow between components in the mesh. When you have configured monitoring, you can grant Prometheus access to scrape the available metrics.

Prerequisites

- You have cluster administrator privileges for your OpenShift Container Platform cluster.
- You have created OpenShift Service Mesh and Knative Serving instances and installed KServe.
- You have downloaded and installed the OpenShift command-line interface (CLI). See Installing the OpenShift CLI.
- You are familiar with creating a config map for monitoring a user-defined workflow. You will perform similar steps in this procedure.
- You are familiar with enabling monitoring for user-defined projects in OpenShift. You will perform similar steps in this procedure.
- You have assigned the **`monitoring-rules-view`** role to users that will monitor metrics.

Procedure

1. In a terminal window, if you are not already logged in to your OpenShift cluster as a cluster administrator, log in to the OpenShift CLI as shown in the following example:

```
$ oc login <openshift_cluster_url> -u <admin_username> -p <password>
```

2. Define a **`ConfigMap`** object in a YAML file called **`uwm-cm-conf.yaml`** with the following contents:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      logLevel: debug
      retention: 15d
```

The **user-workload-monitoring-config** object configures the components that monitor user-defined projects. Observe that the retention time is set to the recommended value of 15 days.

3. Apply the configuration to create the **user-workload-monitoring-config** object.

```
$ oc apply -f uwm-cm-conf.yaml
```

4. Define another **ConfigMap** object in a YAML file called **uwm-cm-enable.yaml** with the following contents:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    enableUserWorkload: true
```

The **cluster-monitoring-config** object enables monitoring for user-defined projects.

5. Apply the configuration to create the **cluster-monitoring-config** object.

```
$ oc apply -f uwm-cm-enable.yaml
```

6. Create **ServiceMonitor** and **PodMonitor** objects to monitor metrics in the service mesh control plane as follows:

   a. Create an **istiod-monitor.yaml** YAML file with the following contents:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: istiod-monitor
  namespace: istio-system
spec:
  targetLabels:
  - app
  selector:
    matchLabels:
      istio: pilot
  endpoints:
  - port: http-monitoring
    interval: 30s
```

   b. Deploy the **ServiceMonitor** CR in the specified **istio-system** namespace.

```
$ oc apply -f istiod-monitor.yaml
```

You see the following output:

```
servicemonitor.monitoring.coreos.com/istiod-monitor created
```

c. Create an **istio-proxies-monitor.yaml** YAML file with the following contents:

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: istio-proxies-monitor
  namespace: istio-system
spec:
  selector:
    matchExpressions:
    - key: istio-prometheus-ignore
      operator: DoesNotExist
  podMetricsEndpoints:
  - path: /stats/prometheus
    interval: 30s
```

d. Deploy the **PodMonitor** CR in the specified **istio-system** namespace.

```
$ oc apply -f istio-proxies-monitor.yaml
```

You see the following output:

```
podmonitor.monitoring.coreos.com/istio-proxies-monitor created
```

3.6. VIEWING METRICS FOR THE SINGLE MODEL SERVING PLATFORM

When a cluster administrator has configured monitoring for the single model serving platform, non-admin users can use the OpenShift web console to view metrics.

Prerequisites

- A cluster administrator has configured monitoring for the single model serving platform.
- You have been assigned the **`monitoring-rules-view`** role.
- You are familiar with how to monitor project metrics in the OpenShift Container Platform web console.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Switch to the **Developer** perspective.
3. In the left menu, click **Observe**.
4. As described in monitoring project metrics, use the web console to run queries for **`caikit_*`**, **`tgi_*`**, **`ovms_*`** or **`istio_*`** metrics.

CHAPTER 4. MONITORING MODEL PERFORMANCE

4.1. VIEWING PERFORMANCE METRICS FOR ALL MODELS ON A MODEL SERVER

In OpenShift AI, you can monitor the following metrics for all the models that are deployed on a model server:

- **HTTP requests** - The number of HTTP requests that have failed or succeeded for all models on the server.

  Note: You can also view the number of HTTP requests that have failed or succeeded for a specific model, as described in Viewing HTTP request metrics for a deployed model.

- **Average response time (ms)** - For all models on the server, the average time it takes the model server to respond to requests.
- **CPU utilization (%)** - The percentage of the CPU's capacity that is currently being used by all models on the server.
- **Memory utilization (%)** - The percentage of the system's memory that is currently being used by all models on the server.

You can specify a time range and a refresh interval for these metrics to help you determine, for example, when the peak usage hours are and how the models are performing at a specified time.

Prerequisites

- You have installed Red Hat OpenShift AI.

- On the OpenShift cluster where OpenShift AI is installed, user workload monitoring is enabled.
- You have logged in to OpenShift AI.
- If you are using specialized OpenShift AI groups, you are part of the user group or admin group (for example, **`rhoai-users`** or **`rhoai-admins`**) in OpenShift.
- There are deployed data science models in your data science project.

Procedure

1. From the OpenShift AI dashboard navigation menu, click **Data Science Projects** and then select the project that contains the data science models that you want to monitor.
2. On the **Components** page, scroll down to the **Models and model servers** section.
3. In the row for the model server that you are interested in, click the action menu (⋮) and then select **View model server metrics**.
4. Optional: On the metrics page for the model server, set the following options:
   - **Time range** - Specifies how long to track the metrics. You can select one of these values: 1 hour, 24 hours, 7 days, and 30 days.
   - **Refresh interval** - Specifies how frequently the graphs on the metrics page are refreshed (to show the latest data). You can select one of these values: 15 seconds, 30 seconds, 1 minute, 5 minutes, 15 minutes, 30 minutes, 1 hour, 2 hours, and 1 day.
5. Scroll down to view data graphs for HTTP requests, average response time, CPU utilization, and memory utilization.

Verification

On the metrics page for the model server, the graphs provide performance metric data.

4.2. VIEWING HTTP REQUEST METRICS FOR A DEPLOYED MODEL

You can view a graph that illustrates the HTTP requests that have failed or succeeded for a specific model.

Prerequisites

- You have installed Red Hat OpenShift AI.
- On the OpenShift cluster where OpenShift AI is installed, user workload monitoring is enabled.
- You have logged in to OpenShift AI.
- If you are using specialized OpenShift AI groups, you are part of the user group or admin group (for example, **`rhoai-users`** or **`rhoai-admins`**) in OpenShift.
- You have deployed a model in a data science project.

Procedure

1. From the OpenShift AI dashboard navigation menu, select **Model Serving**.
2. On the **Model Serving** page, select the model that you are interested in.
3. Optional: On the **Endpoint performance** tab, set the following options:
   - **Time range** - Specifies how long to track the metrics. You can select one of these values: 1 hour, 24 hours, 7 days, and 30 days.
   - **Refresh interval** - Specifies how frequently the graphs on the metrics page are refreshed (to show the latest data). You can select one of these values: 15 seconds, 30 seconds, 1 minute, 5 minutes, 15 minutes, 30 minutes, 1 hour, 2 hours, and 1 day.

Verification

The **Endpoint performance** tab shows a graph of the HTTP metrics for the model.

LEGAL NOTICE