



Red Hat OpenShift AI Self-Managed 2- latest

Serving models

Serve models in Red Hat OpenShift AI Self-Managed

Red Hat OpenShift AI Self-Managed 2-latest Serving models

Serve models in Red Hat OpenShift AI Self-Managed

Legal Notice

Copyright © 2024 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

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

Table of Contents

CHAPTER 1. ABOUT MODEL SERVING	4
CHAPTER 2. SERVING SMALL AND MEDIUM-SIZED MODELS	5
2.1. CONFIGURING MODEL SERVERS	5
2.1.1. Enabling the multi-model serving platform	5
2.1.2. Adding a custom model-serving runtime for the multi-model serving platform	5
2.1.3. Adding a model server for the multi-model serving platform	8
2.1.4. Deleting a model server	9
2.2. WORKING WITH DEPLOYED MODELS	10
2.2.1. Deploying a model by using the multi-model serving platform	10
2.2.2. Viewing a deployed model	12
2.2.3. Updating the deployment properties of a deployed model	12
2.2.4. Deleting a deployed model	14
2.3. CONFIGURING MONITORING FOR THE MULTI-MODEL SERVING PLATFORM	14
2.4. VIEWING MODEL-SERVING RUNTIME METRICS FOR THE MULTI-MODEL SERVING PLATFORM	16
2.5. MONITORING MODEL PERFORMANCE	16
2.5.1. Viewing performance metrics for all models on a model server	16
2.5.2. Viewing HTTP request metrics for a deployed model	17
CHAPTER 3. SERVING LARGE MODELS	19
3.1. ABOUT THE SINGLE-MODEL SERVING PLATFORM	19
3.1.1. Components	19
3.1.2. Installation options	19
3.1.3. Authorization	19
3.1.4. Monitoring	20
3.1.5. Supported model-serving runtimes	20
3.1.6. Inference endpoints	21
3.1.6.1. Example commands	23
3.1.6.2. Additional resources	24
3.2. ABOUT KSERVE DEPLOYMENT MODES	25
3.2.1. Serverless mode	25
3.2.2. Raw deployment mode	25
3.3. CONFIGURING AUTOMATED INSTALLATION OF KSERVE	26
3.4. MANUALLY INSTALLING KSERVE	29
3.4.1. Installing KServe dependencies	30
3.4.1.1. Creating an OpenShift Service Mesh instance	30
3.4.1.2. Creating a Knative Serving instance	32
3.4.1.3. Creating secure gateways for Knative Serving	35
3.4.2. Installing KServe	39
3.4.3. Manually adding an authorization provider	40
3.4.3.1. Installing the Red Hat Authorino Operator	41
3.4.3.2. Creating an Authorino instance	41
3.4.3.3. Configuring an OpenShift Service Mesh instance to use Authorino	43
3.4.3.4. Configuring authorization for KServe	45
3.4.4. Configuring persistent volume claims (PVC) on KServe	47
3.5. ADDING AN AUTHORIZATION PROVIDER FOR THE SINGLE-MODEL SERVING PLATFORM	48
3.6. DEPLOYING MODELS BY USING THE SINGLE-MODEL SERVING PLATFORM	48
3.6.1. Enabling the single-model serving platform	49
3.6.2. Adding a custom model-serving runtime for the single-model serving platform	49
3.6.3. Deploying models on the single-model serving platform	51
3.7. MAKING INFERENCE REQUESTS TO MODELS DEPLOYED ON THE SINGLE-MODEL SERVING PLATFORM	53

3.7.1. Accessing the authorization token for a deployed model	54
3.7.2. Accessing the inference endpoint for a deployed model	54
3.7.2.1. Deploying models on single node openshift using kserve raw deployment mode	55
3.8. CONFIGURING MONITORING FOR THE SINGLE-MODEL SERVING PLATFORM	60
3.9. VIEWING MODEL-SERVING RUNTIME METRICS FOR THE SINGLE-MODEL SERVING PLATFORM	62
3.10. MONITORING MODEL PERFORMANCE	63
3.10.1. Viewing performance metrics for a deployed model	64
3.11. OPTIMIZING MODEL-SERVING RUNTIMES	65
3.11.1. Optimizing the vLLM model-serving runtime	65
3.12. PERFORMANCE TUNING ON THE SINGLE-MODEL SERVING PLATFORM	68
3.12.1. Resolving CUDA out-of-memory errors	68

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 *inference*. 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.
- Your cluster administrator has *not* edited the OpenShift AI dashboard configuration to disable the ability to select the multi-model serving platform, which uses the ModelMesh component. For more information, see [Dashboard configuration options](#).

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 and the model formats supported by those frameworks. By default, the multi-model serving platform includes the OpenVINO Model Server runtime. You can also add your own custom runtime if the default runtime does not meet your needs, such as supporting a specific model format.

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, select **Multi-model serving platform**.



NOTE

The multi-model serving platform supports only the REST protocol. Therefore, you cannot change the default value in the **Select the API protocol this runtime supports** list.

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**
 - a. Click **Upload files**.
 - b. 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**
 - a. Click **Start from scratch**
 - b. 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 custom 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.



NOTE

In OpenShift AI 2-latest, Red Hat supports only NVIDIA GPU accelerators for model serving.

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 NVIDIA GPUs](#).

Procedure

1. In the left menu of the OpenShift AI dashboard, click **Data Science Projects**.
The **Data Science Projects** page opens.
2. Click the name of the project that you want to configure a model server for.
A project details page opens.
3. Click the **Models** tab.
4. 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.

5. In the **Model server name** field, enter a unique name for the model server.
6. 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.

7. In the **Number of model replicas to deploy** field, specify a value.
8. From the **Model server size** list, select a value.
9. 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).
10. 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.
11. 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.
12. 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.
13. Click **Add**.
 - The model server that you configured appears on the **Models** tab for the project, in the **Models and model servers** list.
14. 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 **Models** tab.
4. Click the action menu () beside the project whose model server you want to delete and then click **Delete model server**.
The **Delete model server** dialog opens.
5. Enter the name of the model server in the text field to confirm that you intend to delete it.
6. Click **Delete model server**.

Verification

- The model server that you deleted is no longer displayed on the **Models** tab for the project.

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**. The **Data Science Projects** page opens.
2. Click the name of the project that you want to deploy a model in. A project details page opens.
3. Click the **Models** tab.
4. Click **Deploy model**.
5. 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**
 - i. Select **Existing data connection**
 - ii. From the **Name** list, select a data connection that you previously defined.
 - iii. In the **Path** field, enter the folder path that contains the model in your specified data source.
 - **To use a new data connection**
 - i. To define a new data connection that your model can access, select **New data connection**.
 - ii. In the **Name** field, enter a unique name for the data connection.
 - iii. In the **Access key** field, enter the access key ID for the S3-compatible object storage provider.
 - iv. In the **Secret key** field, enter the secret access key for the S3-compatible object storage account that you specified.

- v. In the **Endpoint** field, enter the endpoint of your S3-compatible object storage bucket.
 - vi. In the **Region** field, enter the default region of your S3-compatible object storage account.
 - vii. In the **Bucket** field, enter the name of your S3-compatible object storage bucket.
 - viii. 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 on the **Models** tab for the 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 model-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 **Deployed models** 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**
 - i. 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**
 - i. In the **Name** field, update a unique name for the data connection.
 - ii. In the **Access key** field, update the access key ID for the S3-compatible object storage provider.
 - iii. In the **Secret key** field, update the secret access key for the S3-compatible object storage account that you specified.
 - iv. In the **Endpoint** field, update the endpoint of your S3-compatible object storage bucket.
 - v. In the **Region** field, update the default region of your S3-compatible object storage account.
 - vi. In the **Bucket** field, update the name of your S3-compatible object storage bucket.
 - vii. 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 of the dashboard.


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 model and model server metrics for the ModelMesh component. ModelMesh generates its own set of metrics and does not rely on the underlying model-serving runtimes to provide them. The set of metrics that ModelMesh generates includes metrics for model request rates and timings, model loading and unloading rates, times and sizes, internal queuing delays, capacity and usage, cache state, and least recently-used models. For more information, see [ModelMesh metrics](#).

After you have configured monitoring, you can view metrics for the ModelMesh component.

Prerequisites

- You have cluster administrator privileges for your OpenShift 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 MODEL-SERVING RUNTIME METRICS FOR THE MULTI-MODEL SERVING PLATFORM

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

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 web console.

Procedure

1. Log in to the OpenShift 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.

2.5. MONITORING MODEL PERFORMANCE

In the multi-model serving platform, you can view performance metrics for all models deployed on a model server and for a specific model that is deployed on the model server.

2.5.1. Viewing performance metrics for all models on a model server

You can monitor the following metrics for all the models that are deployed on a model server:

- **HTTP requests per 5 minutes**- The number of HTTP requests that have failed or succeeded for all models on the server.
- **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.
- You have deployed models on the multi-model serving platform.

Procedure

1. From the OpenShift AI dashboard navigation menu, click **Data Science Projects**. The **Data Science Projects** page opens.
2. Click the name of the project that contains the data science models that you want to monitor.
3. In the project details page, click the **Models** tab.
4. In the row for the model server that you are interested in, click the action menu () and then select **View model server metrics**.
5. 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.
6. Scroll down to view data graphs for HTTP requests per 5 minutes, average response time, CPU utilization, and memory utilization.

Verification

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

2.5.2. 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 that is deployed on the multi-model serving platform.

Prerequisites

- You have installed Red Hat OpenShift AI.
- On the OpenShift cluster where OpenShift AI is installed, user workload monitoring is enabled.
- The following dashboard configuration options are set to the default values as shown:

```
disablePerformanceMetrics:false
disableKServeMetrics:false
```

For more information, see [Dashboard configuration options](#).

- 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 models on the multi-model serving platform.

Procedure

1. From the OpenShift AI dashboard navigation menu, select **Model Serving**.
2. On the **Deployed models** 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.

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

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 on its own model server, the single-model serving platform helps you to deploy, monitor, scale, and maintain large models that require increased resources.

3.1.1. Components

- [KServe](#): A Kubernetes custom resource definition (CRD) that orchestrates model serving for all types of models. KServe includes model-serving runtimes that implement the loading of given types of model servers. KServe also 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.

3.1.2. Installation options

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 configure its dependencies.

For more information about automated installation, see [Configuring automated installation of KServe](#).

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 configure its dependencies. In this situation, you must install KServe manually.

For more information about manual installation, see [Manually installing KServe](#).

3.1.3. Authorization

You can add [Authorino](#) as an authorization provider for the single-model serving platform. Adding an authorization provider allows you to enable token authorization for models that you deploy on the platform, which ensures that only authorized parties can make inference requests to the models.

To add Authorino as an authorization provider on the single-model serving platform, you have the following options:

- If automated installation of the single-model serving platform is possible on your cluster, you can include Authorino as part of the automated installation process.
- If you need to manually install the single-model serving platform, you must also manually configure Authorino.

For guidance on choosing an installation option for the single-model serving platform, see [Installation options](#).

3.1.4. Monitoring

You can configure monitoring for the single-model serving platform and use Prometheus to scrape metrics for each of the pre-installed model-serving runtimes.

3.1.5. Supported model-serving runtimes

OpenShift AI includes several preinstalled model-serving runtimes. You can use preinstalled model-serving runtimes to start serving models without modifying or defining the runtime yourself. You can also add a custom runtime to support a model.

For help adding a custom runtime, see [Adding a custom model-serving runtime for the single-model serving platform](#).

Table 3.1. Model-serving runtimes

Name	Description	Exported model format
Caikit Text Generation Inference Server (Caikit-TGIS) ServingRuntime for KServe (1)	A composite runtime for serving models in the Caikit format	Caikit Text Generation
Caikit Standalone ServingRuntime for KServe (2)	A runtime for serving models in the Caikit embeddings format for embeddings tasks	Caikit Embeddings
OpenVINO Model Server	A scalable, high-performance runtime for serving models that are optimized for Intel architectures	PyTorch, TensorFlow, OpenVINO IR, PaddlePaddle, MXNet, Caffe, Kaldi
Text Generation Inference Server (TGIS) Standalone ServingRuntime for KServe (3)	A runtime for serving TGI-enabled models	PyTorch Model Formats
vLLM ServingRuntime for KServe	A high-throughput and memory-efficient inference and serving runtime for large language models	Supported models

1. 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.

2. The Caikit Standalone runtime is based on [Caikit NLP](#). To use this runtime, you must convert your models to the Caikit embeddings format. For an example, see [Bootstrap Model](#).
3. [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 is incompatible 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](#).

Table 3.2. Deployment requirements

Name	Default protocol	Additional protocol	Model mesh support	Single node OpenShift support	Deployment mode
Caikit Text Generation Inference Server (Caikit-TGIS) ServingRuntime for KServe	REST	gRPC	No	Yes	Raw and serverless
Caikit Standalone ServingRuntime for KServe	REST	gRPC	No	Yes	Raw and serverless
OpenVINO Model Server	REST	None	Yes	Yes	Raw and serverless
Text Generation Inference Server (TGIS) Standalone ServingRuntime for KServe (3)	gRPC	None	No	Yes	Raw and serverless
vLLM ServingRuntime for KServe	REST	None	No	Yes	Raw and serverless

Additional resources

- [Inference endpoints](#)

3.1.6. Inference endpoints

These examples show how to use inference endpoints to query the model.

Caikit TGIS ServingRuntime for KServe

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

Caikit Standalone ServingRuntime for KServe

If you are serving multiple models, you can query **/info/models** or **:443 caikit.runtime.info.InfoService/GetModelsInfo** to view a list of served models.

REST endpoints

- **/api/v1/task/embedding**
- **/api/v1/task/embedding-tasks**
- **/api/v1/task/sentence-similarity**
- **/api/v1/task/sentence-similarity-tasks**
- **/api/v1/task/rerank**
- **/api/v1/task/rerank-tasks**
- **/info/models**
- **/info/version**
- **/info/runtime**

gRPC endpoints

- **:443 caikit.runtime.Nlp.NlpService/EmbeddingTaskPredict**
- **:443 caikit.runtime.Nlp.NlpService/EmbeddingTasksPredict**
- **:443 caikit.runtime.Nlp.NlpService/SentenceSimilarityTaskPredict**
- **:443 caikit.runtime.Nlp.NlpService/SentenceSimilarityTasksPredict**
- **:443 caikit.runtime.Nlp.NlpService/RerankTaskPredict**
- **:443 caikit.runtime.Nlp.NlpService/RerankTasksPredict**
- **:443 caikit.runtime.info.InfoService/GetModelsInfo**
- **:443 caikit.runtime.info.InfoService/GetRuntimeInfo**



NOTE

By default, the Caikit Standalone Runtime exposes REST endpoints. To use gRPC protocol, manually deploy a custom Caikit Standalone ServingRuntime. For more information, see [Adding a custom model-serving runtime for the single-model serving platform](#).

An example manifest is available in the [caikit-tgis-serving GitHub repository](#).

TGIS Standalone ServingRuntime for KServe

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



NOTE

To query the endpoint for the TGIS standalone runtime, you must also download the files in the [proto](#) directory of the OpenShift AI **text-generation-inference** repository.

OpenVINO Model Server

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

vLLM ServingRuntime for KServe

- **:443/version**
- **:443/docs**
- **:443/v1/models**
- **:443/v1/chat/completions**
- **:443/v1/completions**
- **:443/v1/embeddings**
- **:443/tokenize**
- **:443/detokenize**



NOTE

The vLLM runtime is compatible with the OpenAI REST API. For a list of models that the vLLM runtime supports, see [Supported models](#).



NOTE

To use the embeddings inference endpoint in vLLM, you must use an embeddings model that the vLLM supports. You cannot use the embeddings endpoint with generative models. For more information, see [Supported embeddings models in vLLM](#).

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.

3.1.6.1. Example commands

**NOTE**

If you enabled token authorization when deploying the model, add the **Authorization** header and specify a token value.

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 -H 'Authorization:
Bearer <token>'
```

Caikit Standalone ServingRuntime for KServe**REST**

```
curl -H 'Content-Type: application/json' -d '{"inputs": "<text>", "model_id": "<model_id>"}'
<inference_endpoint_url>/api/v1/task/embedding -H 'Authorization: Bearer <token>'
```

gRPC

```
grpcurl -insecure -d '{"text": "<text>"}' -H '\mm-model-id: <model_id>' <inference_endpoint_url>:443
caikit.runtime.Nlp.NlpService/EmbeddingTaskPredict -H 'Authorization: Bearer <token>'
```

TGIS Standalone ServingRuntime for KServe

```
grpcurl -proto text-generation-inference/proto/generation.proto -d '{"requests": [{"text": "<text>"}]}' -H
'Authorization: Bearer <token>' -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> } ]}' -H 'Authorization: Bearer <token>'
```

vLLM ServingRuntime for KServe

```
curl -v https://<inference_endpoint_url>:443/v1/chat/completions -H "Content-Type: application/json" -
d '{"messages": [{"role": "<role>", "content": "<content>"} ]}' -H 'Authorization: Bearer <token>'
```

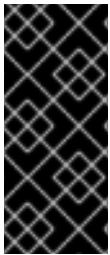
3.1.6.2. Additional resources

- [Text Generation Inference Server \(TGIS\)](#)
- [Caikit API documentation](#)
- [Caikit NLP GitHub project](#)
- [OpenVINO KServe-compatible REST API documentation](#)
- [OpenAI API documentation](#)
- [Supported runtimes](#)

3.2. ABOUT KSERVE DEPLOYMENT MODES

By default, you can deploy models on the single-model serving platform with KServe by using [Red Hat OpenShift Serverless](#), which is a cloud-native development model that allows for serverless deployments of models. OpenShift Serverless is based on the open source [Knative](#) project. In addition, serverless mode is dependent on the Red Hat OpenShift Serverless Operator.

Alternatively, you can use raw deployment mode, which is not dependent on the Red Hat OpenShift Serverless Operator. With raw deployment mode, you can deploy models with Kubernetes resources, such as **Deployment**, **Service**, **Ingress**, and **Horizontal Pod Autoscaler**.



IMPORTANT

Deploying a machine learning model using KServe raw deployment mode is a Limited Availability feature. Limited Availability means that you can install and receive support for the feature only with specific approval from the Red Hat AI Business Unit. Without such approval, the feature is unsupported. In addition, this feature is only supported on Self-Managed deployments of single node OpenShift.

There are both advantages and disadvantages to using each of these deployment modes:

3.2.1. Serverless mode

Advantages:

- Enables autoscaling based on request volume:
 - Resources scale up automatically when receiving incoming requests.
 - Optimizes resource usage and maintains performance during peak times.
- Supports scale down to and from zero using Knative:
 - Allows resources to scale down completely when there are no incoming requests.
 - Saves costs by not running idle resources.

Disadvantages:

- Has customization limitations:
 - Serverless is limited to Knative, such as when mounting multiple volumes.
- Dependency on Knative for scaling:
 - Introduces additional complexity in setup and management compared to traditional scaling methods.

3.2.2. Raw deployment mode

Advantages:

- Enables deployment with Kubernetes resources, such as **Deployment**, **Service**, **Ingress**, and **Horizontal Pod Autoscaler**:

- Provides full control over Kubernetes resources, allowing for detailed customization and configuration of deployment settings.
- Unlocks Knative limitations, such as being unable to mount multiple volumes:
 - Beneficial for applications requiring complex configurations or multiple storage mounts.

Disadvantages:

- Does not support automatic scaling:
 - Does not support automatic scaling down to zero resources when idle.
 - Might result in higher costs during periods of low traffic.
- Requires manual management of scaling.

3.3. 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 configure 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 configure 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 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.
- To add Authorino as an authorization provider so that you can enable token authorization for deployed models, you have installed the **Red Hat - Authorino** Operator. See [Installing the Authorino Operator](#).

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-5fdcf9567c-6cj9d	1/1	Running	0	22h
controller-5fdcf9567c-bf5x7	1/1	Running	0	22h
domain-mapping-56ccd85968-2hjvp	1/1	Running	0	22h
domain-mapping-56ccd85968-ig6mw	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

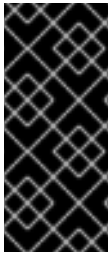
- Verify installation of KServe as follows:
 - In the web console, click **Workloads** → **Pods**.
 - From the project list, select **redhat-ods-applications**. This is the project in which OpenShift AI components are installed, including KServe.
 - Confirm that the project includes a running pod for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

3.4. 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 configure 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.

3.4.1. 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.

3.4.1.1. 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 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

```

3.4.1.2. 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 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
```

- 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

```

- 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
```

- 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 **KnativeService** 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 **KnativeService** 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-5fd9c9567c-6cj9d	1/1	Running	0	22h
controller-5fd9c9567c-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

3.4.1.3. 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 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.

- 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
```

- 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}')
```

- 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}')
```

- 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}
```

- 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
```

- 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
```

- 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
```


- Verify the wildcard certificate.

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

- 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
```

- 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}'
```

- 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
```

- 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

```

3.4.2. 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 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:

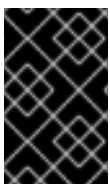
```
spec:
  components:
    kserve:
      managementState: Managed
    serving:
      managementState: Unmanaged
```

- g. Click **Save**.

3.4.3. Manually adding an authorization provider

You can add [Authorino](#) as an authorization provider for the single-model serving platform. Adding an authorization provider allows you to enable token authorization for models that you deploy on the platform, which ensures that only authorized parties can make inference requests to the models.

To manually add Authorino as an authorization provider, you must install the **Red Hat - Authorino** Operator, create an Authorino instance, and then configure the OpenShift Service Mesh and KServe components to use the instance.



IMPORTANT

To manually add an authorization provider, you must make configuration updates to your OpenShift Service Mesh instance. To ensure that your OpenShift Service Mesh instance remains in a supported state, make *only* the updates shown in this section.

Prerequisites

- You have reviewed the options for adding Authorino as an authorization provider and identified manual installation as the appropriate option. See [Adding an authorization provider](#).
- You have manually installed KServe and its dependencies, including OpenShift Service Mesh. See [Manually installing KServe](#).

- When you manually installed KServe, you set the value of the **managementState** field for the **serviceMesh** component to **Unmanaged**. This setting is required for manually adding Authorino. See [Installing KServe](#).

3.4.3.1. Installing the Red Hat Authorino Operator

Before you can add Authorino as an authorization provider, you must install the **Red Hat - Authorino** Operator on your OpenShift cluster.

Prerequisites

- You have cluster administrator privileges for your OpenShift cluster.

Procedure

1. Log in to the OpenShift web console as a cluster administrator.
2. In the web console, click **Operators** → **OperatorHub**.
3. On the **OperatorHub** page, in the **Filter by keyword** field, type **Red Hat - Authorino**.
4. Click the **Red Hat - Authorino** Operator.
5. On the **Red Hat - Authorino** Operator page, review the Operator information and then click **Install**.
6. On the **Install Operator** page, keep the default values for **Update channel**, **Version**, **Installation mode**, **Installed Namespace** and **Update Approval**.
7. Click **Install**.

Verification

- In the OpenShift web console, click **Operators** → **Installed Operators** and confirm that the **Red Hat - Authorino** Operator shows one of the following statuses:
 - **Installing** - installation is in progress; wait for this to change to **Succeeded**. This might take several minutes.
 - **Succeeded** - installation is successful.

3.4.3.2. Creating an Authorino instance

When you have installed the **Red Hat - Authorino** Operator on your OpenShift cluster, you must create an Authorino instance.

Prerequisites

- You have installed the **Red Hat - Authorino** Operator.
- You have privileges to add resources to the project in which your OpenShift Service Mesh instance was created. See [Creating an OpenShift Service Mesh instance](#). For more information about OpenShift permissions, see [Using RBAC to define and apply permissions](#).

Procedure

1. Open a new terminal window.
2. Log in to the OpenShift command-line interface (CLI) as follows:

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

3. Create a namespace to install the Authorino instance.

```
$ oc new-project <namespace_for_authorino_instance>
```



NOTE

The automated installation process creates a namespace called **redhat-ods-applications-auth-provider** for the Authorino instance. Consider using the same namespace name for the manual installation.

4. To enroll the new namespace for the Authorino instance in your existing OpenShift Service Mesh instance, create a new YAML file with the following contents:

```
apiVersion: maistra.io/v1
kind: ServiceMeshMember
metadata:
  name: default
  namespace: <namespace_for_authorino_instance>
spec:
  controlPlaneRef:
    namespace: <namespace_for_service_mesh_instance>
    name: <name_of_service_mesh_instance>
```

5. Save the YAML file.
6. Create the **ServiceMeshMember** resource on your cluster.

```
$ oc create -f <file_name>.yaml
```

7. To configure an Authorino instance, create a new YAML file as shown in the following example:

```
apiVersion: operator.authorino.kuadrant.io/v1beta1
kind: Authorino
metadata:
  name: authorino
  namespace: <namespace_for_authorino_instance>
spec:
  authConfigLabelSelectors: security.opendatahub.io/authorization-group=default
  clusterWide: true
  listener:
    tls:
      enabled: false
  oidcServer:
    tls:
      enabled: false
```

8. Save the YAML file.
9. Create the **Authorino** resource on your cluster.

```
$ oc create -f <file_name>.yaml
```

10. Patch the Authorino deployment to inject an Istio sidecar, which makes the Authorino instance part of your OpenShift Service Mesh instance.

```
$ oc patch deployment <name_of_authorino_instance> -n
<namespace_for_authorino_instance> -p '{"spec": {"template":{"metadata":{"labels":
{"sidecar.istio.io/inject":"true"}}}} }'
```

Verification

- Confirm that the Authorino instance is running as follows:
 1. Check the pods (and containers) that are running in the namespace that you created for the Authorino instance, as shown in the following example:

```
$ oc get pods -n redhat-ods-applications-auth-provider -o="custom-
columns=NAME:.metadata.name,STATUS:.status.phase,CONTAINERS:.spec.containers[*
].name"
```

2. Confirm that the output resembles the following example:

```
NAME                STATUS  CONTAINERS
authorino-6bc64bd667-kn28z  Running  authorino,istio-proxy
```

As shown in the example, there is a single running pod for the Authorino instance. The pod has containers for Authorino and for the Istio sidecar that you injected.

3.4.3.3. Configuring an OpenShift Service Mesh instance to use Authorino

When you have created an Authorino instance, you must configure your OpenShift Service Mesh instance to use Authorino as an authorization provider.



IMPORTANT

To ensure that your OpenShift Service Mesh instance remains in a supported state, make *only* the configuration updates shown in the following procedure.

Prerequisites

- You have created an Authorino instance and enrolled the namespace for the Authorino instance in your OpenShift Service Mesh instance.
- You have privileges to modify the OpenShift Service Mesh instance. See [Creating an OpenShift Service Mesh instance](#).

Procedure

1. In a terminal window, if you are not already logged in to your OpenShift cluster as a user that has privileges to update the OpenShift Service Mesh instance, log in to the OpenShift CLI as shown in the following example:

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

2. Create a new YAML file with the following contents:

```
spec:
  techPreview:
    meshConfig:
      extensionProviders:
        - name: redhat-ods-applications-auth-provider
      envoyExtAuthzGrpc:
        service: <name_of_authorino_instance>-authorino-
        authorization.<namespace_for_authorino_instance>.svc.cluster.local
        port: 50051
```

3. Save the YAML file.
4. Use the **oc patch** command to apply the YAML file to your OpenShift Service Mesh instance.

```
$ oc patch smcp <name_of_service_mesh_instance> --type merge -n
<namespace_for_service_mesh_instance> --patch-file <file_name>.yaml
```



IMPORTANT

You can apply the configuration shown as a patch *only* if you have not already specified other extension providers in your OpenShift Service Mesh instance. If you have already specified other extension providers, you must manually edit your **ServiceMeshControlPlane** resource to add the configuration.

Verification

- Verify that your Authorino instance has been added as an extension provider in your OpenShift Service Mesh configuration as follows:
 1. Inspect the **ConfigMap** object for your OpenShift Service Mesh instance:

```
$ oc get configmap istio-<name_of_service_mesh_instance> -n
<namespace_for_service_mesh_instance> --output=jsonpath={.data.mesh}
```

2. Confirm that you see output similar to the following example, which shows that the Authorino instance has been successfully added as an extension provider.

```
defaultConfig:
  discoveryAddress: istiod-data-science-smcp.istio-system.svc:15012
  proxyMetadata:
    ISTIO_META_DNS_AUTO_ALLOCATE: "true"
    ISTIO_META_DNS_CAPTURE: "true"
    PROXY_XDS_VIA_AGENT: "true"
  terminationDrainDuration: 35s
  tracing: {}
  dnsRefreshRate: 300s
```



```

enablePrometheusMerge: true
extensionProviders:
- envoyExtAuthzGrpc:
  port: 50051
  service: authorino-authorino-authorization.opendatahub-auth-provider.svc.cluster.local
  name: opendatahub-auth-provider
ingressControllerMode: "OFF"
rootNamespace: istio-system
trustDomain: null%

```

3.4.3.4. Configuring authorization for KServe

To configure the single-model serving platform to use Authorino, you must create a global **AuthorizationPolicy** resource that is applied to the KServe predictor pods that are created when you deploy a model. In addition, to account for the multiple network hops that occur when you make an inference request to a model, you must create an **EnvoyFilter** resource that continually resets the HTTP host header to the one initially included in the inference request.

Prerequisites

- You have created an Authorino instance and configured your OpenShift Service Mesh to use it.
- You have privileges to update the KServe deployment on your cluster.
- You have privileges to add resources to the project in which your OpenShift Service Mesh instance was created. See [Creating an OpenShift Service Mesh instance](#).

Procedure

1. In a terminal window, if you are not already logged in to your OpenShift cluster as a user that has privileges to update the KServe deployment, log in to the OpenShift CLI as shown in the following example:

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

2. Create a new YAML file with the following contents:

```

apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: kserve-predictor
spec:
  action: CUSTOM
  provider:
    name: redhat-ods-applications-auth-provider 1
  rules:
    - to:
      - operation:
          notPaths:
            - /healthz
            - /debug/pprof/
            - /metrics
            - /wait-for-drain

```

```

selector:
  matchLabels:
    component: predictor

```

- 1 The name that you specify must match the name of the extension provider that you added to your OpenShift Service Mesh instance.

3. Save the YAML file.
4. Create the **AuthorizationPolicy** resource in the namespace for your OpenShift Service Mesh instance.

```
$ oc create -n <namespace_for_service_mesh_instance> -f <file_name>.yaml
```

5. Create another new YAML file with the following contents:

```

apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: activator-host-header
spec:
  priority: 20
  workloadSelector:
    labels:
      component: predictor
  configPatches:
  - applyTo: HTTP_FILTER
    match:
      listener:
        filterChain:
          filter:
            name: envoy.filters.network.http_connection_manager
    patch:
      operation: INSERT_BEFORE
      value:
        name: envoy.filters.http.lua
        typed_config:
          '@type': type.googleapis.com/envoy.extensions.filters.http.lua.v3.Lua
          inlineCode: |
            function envoy_on_request(request_handle)
              local headers = request_handle:headers()
              if not headers then
                return
              end
              local original_host = headers:get("k-original-host")
              if original_host then
                port_seperator = string.find(original_host, ":", 7)
                if port_seperator then
                  original_host = string.sub(original_host, 0, port_seperator-1)
                end
                headers:replace('host', original_host)
              end
            end
          end

```

The **EnvoyFilter** resource shown continually resets the HTTP host header to the one initially included in any inference request.

6. Create the **EnvoyFilter** resource in the namespace for your OpenShift Service Mesh instance.

```
$ oc create -n <namespace_for_service_mesh_instance> -f <file_name>.yaml
```

Verification

- Check that the **AuthorizationPolicy** resource was successfully created.

```
$ oc get authorizationpolicies -n <namespace_for_service_mesh_instance>
```

Confirm that you see output similar to the following example:

```
NAME          AGE
kserve-predictor 28h
```

- Check that the **EnvoyFilter** resource was successfully created.

```
$ oc get envoyfilter -n <namespace_for_service_mesh_instance>
```

Confirm that you see output similar to the following example:

```
NAME                AGE
activator-host-header 28h
```

3.4.4. Configuring persistent volume claims (PVC) on KServe

Enable persistent volume claims (PVC) on your inference service so you can provision persistent storage. For more information about PVC, see [Understanding persistent storage](#).

To enable PVC, from the OpenShift AI dashboard, select the **Project** dropdown and click **knative-serving**. Then, follow the steps in [Enabling PVC support](#).

Verification

Verify that the inference service allows PVC as follows:

- In the OpenShift web console, change into the **Administrator** perspective.
- Click **Home** → **Search**.
- In **Resources**, search for **InferenceService**.
- Click the name of the inference service.
- Click the **YAML** tab.
- Confirm that **volumeMounts** appears, similar to the following output:

```
apiVersion: "serving.kserve.io/v1beta1"
kind: "InferenceService"
metadata:
```

```

name: "sklearn-iris"
spec:
  predictor:
    model:
      runtime: kserve-mlserver
      modelFormat:
        name: sklearn
      storageUri: "gs://kfserving-examples/models/sklearn/1.0/model"
      volumeMounts:
        - name: my-dynamic-volume
          mountPath: /tmp/data
    volumes:
      - name: my-dynamic-volume
        persistentVolumeClaim:
          claimName: my-dynamic-pvc

```

3.5. ADDING AN AUTHORIZATION PROVIDER FOR THE SINGLE-MODEL SERVING PLATFORM

You can add [Authorino](#) as an authorization provider for the single-model serving platform. Adding an authorization provider allows you to enable token authorization for models that you deploy on the platform, which ensures that only authorized parties can make inference requests to the models.

The method that you use to add Authorino as an authorization provider depends on how you install the single-model serving platform. The installation options for the platform are described as follows:

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. You can include Authorino as part of the automated installation process.

For more information about automated installation, including Authorino, see [Configuring automated installation of KServe](#).

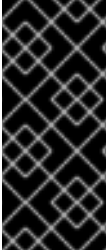
Manual installation

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. You must also manually configure Authorino.

For more information about manual installation, including Authorino, see [Manually installing KServe](#).

3.6. DEPLOYING MODELS BY USING THE SINGLE-MODEL SERVING PLATFORM

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



IMPORTANT

If you want to use the single-model serving platform to deploy a model from S3-compatible storage that uses a self-signed SSL certificate, you must install a certificate authority (CA) bundle on your OpenShift cluster. For more information, see [Working with certificates](#) (OpenShift AI Self-Managed) or [Working with certificates](#) (OpenShift AI Self-Managed in a disconnected environment).

3.6.1. 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.
- Your cluster administrator has *not* edited the OpenShift AI dashboard configuration to disable the ability to select the single-model serving platform, which uses the KServe component. For more information, see [Dashboard configuration options](#).

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 preinstalled 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 preinstalled runtimes and any custom runtimes that you have added.

For more information about preinstalled runtimes, see [Supported runtimes](#).
 - b. Set the runtime that you want to use to **Enabled**.

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

3.6.2. 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 and the model formats supported by those frameworks. You can use the [pre-installed runtimes](#) that are included with

OpenShift AI. You can also add your own custom runtimes if the default runtimes do not meet your needs. For example, if the TGIS runtime does not support a model format that is supported by [Hugging Face Text Generation Inference \(TGI\)](#), 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

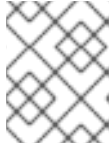
OpenShift AI enables you to add your own custom runtimes, but does not support the runtimes themselves. You are responsible for correctly configuring and maintaining custom runtimes. You are also responsible for ensuring that you are licensed to use any custom runtimes that you add.

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, select **Single-model serving platform**.
4. In the **Select the API protocol this runtime supports** list, select **REST** or **gRPC**.
5. 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**
 - a. Click **Upload files**.
 - b. 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**
 - a. Click **Start from scratch**
 - b. 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.

6. Click **Add**.
The **Serving runtimes** page opens and shows the updated list of runtimes that are installed. Observe that the custom runtime that you added is automatically enabled. The API protocol that you specified when creating the runtime is shown.
7. Optional: To edit your custom runtime, click the action menu (**:**) and select **Edit**.

Verification

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

3.6.3. 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.
- 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.
- 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.
- If you want to use graphics processing units (GPUs) with your model server, you have enabled GPU support in OpenShift AI. See [Enabling NVIDIA GPUs](#).

- To use the vLLM runtime, you have enabled GPU support in OpenShift AI and have installed and configured the Node Feature Discovery operator on your cluster. For more information, see [Installing the Node Feature Discovery operator](#) and [Enabling NVIDIA GPUs](#)



NOTE

In OpenShift AI 2-latest, Red Hat supports only NVIDIA GPU accelerators for model serving.

Procedure

1. In the left menu, click **Data Science Projects**.
The **Data Science Projects** page opens.
2. Click the name of the project that you want to deploy a model in.
A project details page opens.
3. Click the **Models** tab.
4. 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.

5. In the **Model name** field, enter a unique name for the model that you are deploying.
6. In the **Serving runtime** field, select an enabled runtime.
7. From the **Model framework** list, select a value.
8. In the **Number of model replicas to deploy** field, specify a value.
9. From the **Model server size** list, select a value.
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. To require token authorization for inference requests to the deployed model, perform the following actions:
 - a. Select **Require token authorization**.
 - b. In the **Service account name** field, enter the service account name that the token will be generated for.
12. To specify the location of your model, perform one of the following sets of actions:
 - **To use an existing data connection**
 - a. Select **Existing data connection**
 - b. From the **Name** list, select a data connection that you previously defined.

- c. In the **Path** field, enter the folder path that contains the model in your specified data source.



IMPORTANT

The OpenVINO Model Server runtime has specific requirements for how you specify the model path. For more information, see known issue [RHOAIENG-3025](#) in the OpenShift AI release notes.

- **To use a new data connection**

- a. To define a new data connection that your model can access, select **New data connection**.
- b. In the **Name** field, enter a unique name for the data connection.
- c. In the **Access key** field, enter the access key ID for your S3-compatible object storage provider.
- d. In the **Secret key** field, enter the secret access key for the S3-compatible object storage account that you specified.
- e. In the **Endpoint** field, enter the endpoint of your S3-compatible object storage bucket.
- f. In the **Region** field, enter the default region of your S3-compatible object storage account.
- g. In the **Bucket** field, enter the name of your S3-compatible object storage bucket.
- h. In the **Path** field, enter the folder path in your S3-compatible object storage that contains your data file.



IMPORTANT

The OpenVINO Model Server runtime has specific requirements for how you specify the model path. For more information, see known issue [RHOAIENG-3025](#) in the OpenShift AI release notes.

13. Click **Deploy**.

Verification

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

3.7. MAKING INFERENCE REQUESTS TO 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 the inference endpoint for the model.

In addition, if you secured your inference endpoint by enabling token authorization, you must know how to access your authorization token so that you can specify this in your inference requests.


3.7.1. Accessing the authorization token for a deployed model

If you secured your model inference endpoint by enabling token authorization, you must know how to access your authorization token so that you can specify it in your inference requests.

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 **Data Science Projects**.
The **Data Science Projects** page opens.
2. Click the name of the project that contains your deployed model.
A project details page opens.
3. Click the **Models** tab.
4. In the **Models and model servers** list, expand the section for your model.
Your authorization token is shown in the **Token authorization** section, in the **Token secret** field.
5. Optional: To copy the authorization token for use in an inference request, click the **Copy** button () next to the token value.

3.7.2. Accessing the inference endpoint for a deployed model

To make inference requests to your deployed model, you must know how to access the inference endpoint that is 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.
- If you enabled token authorization for your deployed model, you have the associated token value.

Procedure

1. From the OpenShift AI dashboard, click **Model Serving**.
The inference endpoint for the model is shown in the **Inference endpoint** field.

- Depending on what action you want to perform with the model (and if the model supports that action), copy the inference endpoint shown and then add a path to the end of the URL.

**NOTE**

For a list of paths to use with the supported runtimes, see [Inference endpoints](#).

- Use the endpoint to make API requests to your deployed model.

**NOTE**

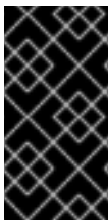
For a list of sample commands, see [Inference endpoints](#).

Additional resources

- [Text Generation Inference Server \(TGIS\)](#)
- [Caikit API documentation](#)
- [Caikit Text Embedding GitHub project](#)
- [OpenVINO KServe-compatible REST API documentation](#)
- [OpenAI API documentation](#)

3.7.2.1. Deploying models on single node openshift using kserve raw deployment mode

You can deploy a machine learning model by using KServe raw deployment mode on single node OpenShift. Raw deployment mode offers several advantages over Knative, such as the ability to mount multiple volumes.

**IMPORTANT**

Deploying a machine learning model using KServe raw deployment mode on single node OpenShift is a Limited Availability feature. Limited Availability means that you can install and receive support for the feature only with specific approval from the Red Hat AI Business Unit. Without such approval, the feature is unsupported.

Prerequisites

- You have logged in to Red Hat OpenShift AI.
- You have cluster administrator privileges for your OpenShift cluster.
- You have created an OpenShift cluster that has a node with at least 4 CPUs and 16 GB memory.
- You have installed the Red Hat OpenShift AI (RHOAI) Operator.
- You have installed the OpenShift command-line interface (CLI). For more information about installing the OpenShift command-line interface (CLI), see [Getting started with the OpenShift CLI](#).
- You have installed KServe.
- 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.
- 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.
- 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](#).
- To use the vLLM runtime, you have enabled GPU support in OpenShift AI and have installed and configured the Node Feature Discovery operator on your cluster. For more information, see [Installing the Node Feature Discovery operator](#) and [Enabling GPU support in OpenShift AI](#)

Procedure

1. Open a command-line terminal and log in to your OpenShift cluster as cluster administrator:

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

2. By default, OpenShift uses a service mesh for network traffic management. As KServe raw deployment mode does not require a service mesh, disable Red Hat OpenShift Service Mesh:

- a. Enter the following command to disable Red Hat OpenShift Service Mesh:

```
$ oc edit dsci -n redhat-ods-operator
```

- b. In the YAML editor, change the value of **managementState** for the **serviceMesh** component to **Removed** as shown:

```
spec:
  components:
    serviceMesh:
      managementState: Removed
```

- c. Save the changes.

3. Create a project:

```
$ oc new-project <project_name> --description="<description>" --display-name="<display_name>"
```

For information about creating projects, see [Working with projects](#).

4. Create a data science cluster:

- a. In the Red Hat OpenShift web console **Administrator** view, click **Operators** → **Installed Operators** and then click the Red Hat OpenShift AI Operator.
- b. Click the **Data Science Cluster** tab.
- c. Click the **Create DataScienceCluster** button.
- d. In the **Configure via** field, click the **YAML view** radio button.

- e. In the **spec.components** section of the YAML editor, configure the **kserve** component as shown:

```
kserve:
  defaultDeploymentMode: RawDeployment
  managementState: Managed
  serving:
    managementState: Removed
    name: knative-serving
```

- f. Click **Create**.

5. Create a secret file:

- a. At your command-line terminal, create a YAML file to contain your secret and add the following YAML code:

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    serving.kserve.io/s3-endpoint: <AWS_ENDPOINT>
    serving.kserve.io/s3-usehttps: "1"
    serving.kserve.io/s3-region: <AWS_REGION>
    serving.kserve.io/s3-useanoncredential: "false"
  name: <Secret-name>
stringData:
  AWS_ACCESS_KEY_ID: "<AWS_ACCESS_KEY_ID>"
  AWS_SECRET_ACCESS_KEY: "<AWS_SECRET_ACCESS_KEY>"
```



IMPORTANT

If you are deploying a machine learning model in a disconnected deployment, add **serving.kserve.io/s3-verifyssl: '0'** to the **metadata.annotations** section.

- b. Save the file with the file name **secret.yaml**.
- c. Apply the **secret.yaml** file:

```
$ oc apply -f secret.yaml -n <namespace>
```

6. Create a service account:

- a. Create a YAML file to contain your service account and add the following YAML code:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: models-bucket-sa
secrets:
  - name: s3creds
```

For information about service accounts, see [Understanding and creating service accounts](#).

b. Save the file with the file name **serviceAccount.yaml**.

c. Apply the **serviceAccount.yaml** file:

```
$ oc apply -f serviceAccount.yaml -n <namespace>
```

7. Create a YAML file for the serving runtime to define the container image that will serve your model predictions. Here is an example using the OpenVino Model Server:

```
apiVersion: serving.kserve.io/v1alpha1
kind: ServingRuntime
metadata:
  name: ovms-runtime
spec:
  annotations:
    prometheus.io/path: /metrics
    prometheus.io/port: "8888"
  containers:
    - args:
      - --model_name={{.Name}}
      - --port=8001
      - --rest_port=8888
      - --model_path=/mnt/models
      - --file_system_poll_wait_seconds=0
      - --grpc_bind_address=0.0.0.0
      - --rest_bind_address=0.0.0.0
      - --target_device=AUTO
      - --metrics_enable
      image:
        quay.io/modh/openvino_model_server@sha256:6c7795279f9075bebfcd9aecbb4a4ce4177eec4
        1fb3f3e1f1079ce6309b7ae45
        name: kserve-container
      ports:
        - containerPort: 8888
          protocol: TCP
    multiModel: false
  protocolVersions:
    - v2
    - grpc-v2
  supportedModelFormats:
    - autoSelect: true
      name: openvino_ir
      version: opset13
    - name: onnx
      version: "1"
    - autoSelect: true
      name: tensorflow
      version: "1"
    - autoSelect: true
      name: tensorflow
      version: "2"
    - autoSelect: true
      name: paddle
      version: "2"
```

```
- autoSelect: true
  name: pytorch
  version: "2"
```

- a. If you are using the OpenVINO Model Server example above, ensure that you insert the correct values required for any placeholders in the YAML code.
- b. Save the file with an appropriate file name.
- c. Apply the file containing your serving run time:

```
$ oc apply -f <serving run time file name> -n <namespace>
```

8. Create an InferenceService custom resource (CR). Create a YAML file to contain the InferenceService CR. Using the OpenVINO Model Server example used previously, here is the corresponding YAML code:

```
apiVersion: serving.kserve.io/v1beta1
kind: InferenceService
metadata:
  annotations:
    serving.knative.openshift.io/enablePassthrough: "true"
    sidecar.istio.io/inject: "true"
    sidecar.istio.io/rewriteAppHTTPProbers: "true"
    serving.kserve.io/deploymentMode: RawDeployment
  name: <InferenceService-Name>
spec:
  predictor:
    scaleMetric:
    minReplicas: 1
    scaleTarget:
    canaryTrafficPercent:
    serviceAccountName: <serviceAccountName>
  model:
    env: []
    volumeMounts: []
    modelFormat:
      name: onnx
    runtime: ovms-runtime
    storageUri: s3://<bucket_name>/<model_directory_path>
    resources:
      requests:
        memory: 5Gi
    volumes: []
```

- a. In your YAML code, ensure the following values are set correctly:
 - **serving.kserve.io/deploymentMode** must contain the value **RawDeployment**.
 - **modelFormat** must contain the value for your model format, such as **onnx**.
 - **storageUri** must contain the value for your model s3 storage directory, for example **s3://<bucket_name>/<model_directory_path>**.
 - **runtime** must contain the value for the name of your serving runtime, for example, **ovms-runtime**.

- b. Save the file with an appropriate file name.
- c. Apply the file containing your InferenceService CR:

```
$ oc apply -f <InferenceService CR file name> -n <namespace>
```

9. Verify that all pods are running in your cluster:

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

Example output:

```
NAME READY STATUS RESTARTS AGE
<isvc_name>-predictor-xxxxx-2mr5l 1/1 Running 2 165m
console-698d866b78-m87pm 1/1 Running 2 165m
```

10. After you verify that all pods are running, forward the service port to your local machine:

```
$ oc -n <namespace> port-forward pod/<pod-name> <local_port>:<remote_port>
```

Ensure that you replace **<namespace>**, **<pod-name>**, **<local_port>**, **<remote_port>** (this is the model server port, for example, **8888**) with values appropriate to your deployment.

Verification

- Use your preferred client library or tool to send requests to the **localhost** inference URL.

3.8. CONFIGURING MONITORING FOR THE SINGLE-MODEL SERVING PLATFORM

The single-model serving platform includes metrics for [supported runtimes](#) of the KServe component. KServe does not generate its own metrics and relies on the underlying model-serving runtimes to provide them. The set of available metrics for a deployed model depends on its model-serving runtime.

In addition to runtime metrics for KServe, you can also configure monitoring for OpenShift Service Mesh. The OpenShift Service Mesh metrics help you to understand dependencies and traffic flow between components in the mesh.

Prerequisites

- You have cluster administrator privileges for your OpenShift 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:

```
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
```

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:

```
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
```

3.9. VIEWING MODEL-SERVING RUNTIME 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 model-serving runtime metrics for the KServe component.

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 web console.

Procedure

1. Log in to the OpenShift 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_*** and **vllm:*** model-serving runtime metrics. You can also run queries for **istio_*** metrics that are related to OpenShift Service Mesh. Some examples are shown.
 - a. The following query displays the number of successful inference requests over a period of time for a model deployed with the vLLM runtime:

```
sum(increase(vllm:request_success_total{namespace=${namespace},model_name=${model_name}}[${rate_interval}]))
```

- b. The following query displays the number of successful inference requests over a period of time for a model deployed with the standalone TGIS runtime:

```
sum(increase(tgi_request_success{namespace=${namespace},pod=~${model_name}-predictor-.*}${rate_interval}))
```

- c. The following query displays the number of successful inference requests over a period of time for a model deployed with the Caikit Standalone runtime:

```
sum(increase(predict_rpc_count_total{namespace=${namespace},code=OK,model_id=${model_name}}[${rate_interval}]))
```

- d. The following query displays the number of successful inference requests over a period of time for a model deployed with the OpenVINO Model Server runtime:

```
sum(increase(ovms_requests_success{namespace=${namespace},name=${model_name}}[${rate_interval}]))
```

Additional resources

- [OVMS metrics](#)
- [TGIS metrics](#)
- [vLLM metrics](#)

3.10. MONITORING MODEL PERFORMANCE

In the single-model serving platform, you can view performance metrics for a specific model that is deployed on the platform.

3.10.1. Viewing performance metrics for a deployed model

You can monitor the following metrics for a specific model that is deployed on the single-model serving platform:

- **Number of requests** - The number of requests that have failed or succeeded for a specific model.
- **Average response time (ms)** - The average time it takes a specific model to respond to requests.
- **CPU utilization (%)** - The percentage of the CPU limit per model replica that is currently utilized by a specific model.
- **Memory utilization (%)** - The percentage of the memory limit per model replica that is utilized by a specific model.

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 model is performing at a specified time.

Prerequisites

- You have installed Red Hat OpenShift AI.
- A cluster admin has enabled user workload monitoring (UWM) for user-defined projects on your OpenShift cluster. For more information, see [Enabling monitoring for user-defined projects](#) and [Configuring monitoring for the single-model serving platform](#).
- 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.
- The following dashboard configuration options are set to the default values as shown:

```
disablePerformanceMetrics:false
disableKServeMetrics:false
```

For more information, see [Dashboard configuration options](#).

- You have deployed a model on the single-model serving platform by using a preinstalled runtime.



NOTE

Metrics are only supported for models deployed by using a preinstalled model-serving runtime or a custom runtime that is duplicated from a preinstalled runtime.

Procedure

1. From the OpenShift AI dashboard navigation menu, click **Data Science Projects**. The **Data Science Projects** page opens.
2. Click the name of the project that contains the data science models that you want to monitor.

3. In the project details page, click the **Models** tab.
4. Select the model that you are interested in.
5. 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.
6. Scroll down to view data graphs for number of requests, average response time, CPU utilization, and memory utilization.

Verification

The **Endpoint performance** tab shows graphs of metrics for the model.

3.11. OPTIMIZING MODEL-SERVING RUNTIMES

You can optionally enhance the preinstalled model-serving runtimes available in OpenShift AI to leverage additional benefits and capabilities, such as optimized inferencing, reduced latency, and fine-tuned resource allocation.

3.11.1. Optimizing the vLLM model-serving runtime

You can configure the **vLLM ServingRuntime for KServer** runtime to use speculative decoding, a parallel processing technique to optimize inferencing time for large language models (LLMs).

You can also configure the runtime to support inferencing for vision-language models (VLMs). VLMs are a subset of multi-modal models that integrate both visual and textual data.

To configure the **vLLM ServingRuntime for KServer** runtime for speculative decoding or multi-modal inferencing, you must add additional arguments in the vLLM model-serving runtime.

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, **oai-admin-group**) in OpenShift.
- If you used the pre-installed **vLLM ServingRuntime for KServer** runtime, you duplicated the runtime to create a custom version. For more information about duplicating the pre-installed vLLM runtime, see [Adding a custom model-serving runtime for the single-model serving platform](#).
- If you are using the vLLM model-serving runtime for speculative decoding with a draft model, you have stored the original model and the speculative model in the same folder within your S3-compatible object storage.

Procedure

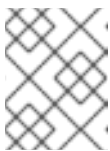
1. From the OpenShift AI dashboard, click **Settings** > **Serving runtimes**.

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

2. Find the custom vLLM model-serving runtime that you created, click the action menu (:) next to the runtime and select **Edit**.
The embedded YAML editor opens and shows the contents of the custom model-serving runtime.
3. To configure the vLLM model-serving runtime for speculative decoding by matching n-grams in the prompt:
 - a. Add the following arguments:

```
containers:
  - args:
    - --speculative-model=[ngram]
    - --num-speculative-tokens=<NUM_SPECULATIVE_TOKENS>
    - --ngram-prompt-lookup-max=<NGRAM_PROMPT_LOOKUP_MAX>
    - --use-v2-block-manager
```

- b. Replace **<NUM_SPECULATIVE_TOKENS>** and **<NGRAM_PROMPT_LOOKUP_MAX>** with your own values.



NOTE

Inferencing throughput varies depending on the model used for speculating with n-grams.

4. To configure the vLLM model-serving runtime for speculative decoding with a draft model:
 - a. Remove the **--model** argument:

```
containers:
  - args:
    - --model=/mnt/models
```

- b. Add the following arguments:

```
containers:
  - args:
    - --port=8080
    - --served-model-name={{.Name}}
    - --distributed-executor-backend=mp
    - --model=/mnt/models/<path_to_original_model>
    - --speculative-model=/mnt/models/<path_to_speculative_model>
    - --num-speculative-tokens=<NUM_SPECULATIVE_TOKENS>
    - --use-v2-block-manager
```

- c. Replace **<path_to_speculative_model>** and **<path_to_original_model>** with the paths to the speculative model and original model on your S3-compatible object storage.
 - d. Replace **<NUM_SPECULATIVE_TOKENS>** with your own value.
5. To configure the vLLM model-serving runtime for multi-modal inferencing:
 - a. Add the following arguments:

- a. Add the following arguments.

```
containers:
  - args:
    - --trust-remote-code
```



NOTE

Only use the **--trust-remote-code** argument with models from trusted sources.

6. Click **Update**.
The **Serving runtimes** page opens and shows the list of runtimes that are installed. Confirm that the custom model-serving runtime you updated is shown.
7. Deploy the model by using the custom runtime as described in [Deploying models on the single-model serving platform](#).

Verification

- If you have configured the vLLM model-serving runtime for speculative decoding, use the following example command to verify API requests to your deployed model:

```
curl -v https://<inference_endpoint_url>:443/v1/chat/completions
-H "Content-Type: application/json"
-H "Authorization: Bearer <token>"
```

- If you have configured the vLLM model-serving runtime for multi-modal inferencing, use the following example command to verify API requests to the vision-language model (VLM) that you have deployed:

```
curl -v https://<inference_endpoint_url>:443/v1/chat/completions
-H "Content-Type: application/json"
-H "Authorization: Bearer <token>"
-d '{"model": "<model_name>",
  "messages":
    [{"role": "<role>",
      "content":
        [{"type": "text", "text": "<text>"
        },
        {"type": "image_url", "image_url": "<image_url_link>"
        }
      ]
    }
  ]
}'
```

Additional resources

- [vLLM Engine Arguments](#)
- [OpenAI Compatible Server](#)

3.12. PERFORMANCE TUNING ON THE SINGLE-MODEL SERVING PLATFORM

Certain performance issues might require you to tune the parameters of your inference service or model-serving runtime.

3.12.1. Resolving CUDA out-of-memory errors

In certain cases, depending on the model and hardware accelerator used, the TGIS memory auto-tuning algorithm might underestimate the amount of GPU memory needed to process long sequences. This miscalculation can lead to Compute Unified Architecture (CUDA) out-of-memory (OOM) error responses from the model server. In such cases, you must update or add additional parameters in the TGIS model-serving runtime, as described in the following procedure.

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. 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. Based on the runtime that you used to deploy your model, perform one of the following actions:
 - If you used the pre-installed **TGIS Standalone ServingRuntime for KServe** runtime, duplicate the runtime to create a custom version and then follow the remainder of this procedure. For more information about duplicating the pre-installed TGIS runtime, see [Adding a custom model-serving runtime for the single-model serving platform](#).
 - If you were already using a custom TGIS runtime, click the action menu (**:**) next to the runtime and select **Edit**.
The embedded YAML editor opens and shows the contents of the custom model-serving runtime.
3. Add or update the **BATCH_SAFETY_MARGIN** environment variable and set the value to 30. Similarly, add or update the **ESTIMATE_MEMORY_BATCH_SIZE** environment variable and set the value to 8.

```
spec:
  containers:
    env:
      - name: BATCH_SAFETY_MARGIN
        value: 30
      - name: ESTIMATE_MEMORY_BATCH_SIZE
        value: 8
```


**NOTE**

The **BATCH_SAFETY_MARGIN** parameter sets a percentage of free GPU memory to hold back as a safety margin to avoid OOM conditions. The default value of **BATCH_SAFETY_MARGIN** is **20**. The **ESTIMATE_MEMORY_BATCH_SIZE** parameter sets the batch size used in the memory auto-tuning algorithm. The default value of **ESTIMATE_MEMORY_BATCH_SIZE** is **16**.

4. Click **Update**.

The **Serving runtimes** page opens and shows the list of runtimes that are installed. Observe that the custom model-serving runtime you updated is shown.

5. To redeploy the model for the parameter updates to take effect, perform the following actions:

- a. From the OpenShift AI dashboard, click **Model Serving** > **Deployed Models**
- b. Find the model you want to redeploy, click the action menu (:) next to the model, and select **Delete**.
- c. Redeploy the model as described in [Deploying models on the single-model serving platform](#).

Verification

- You receive successful responses from the model server and no longer see CUDA OOM errors.