Red Hat OpenShift AI Self-Managed 2.16

Monitoring data science models

Monitor your OpenShift AI models for fairness

Last Updated: 2025-02-03

Red Hat OpenShift AI Self-Managed 2.16 Monitoring data science models

Monitor your OpenShift AI models for fairness

Legal Notice

Abstract

Monitor your machine-learning models in OpenShift AI to ensure they are transparent, fair, and reliable.

Table of Contents

CHAPTER 1. OVERVIEW OF MODEL MONITORING

To ensure that machine-learning models are transparent, fair, and reliable, data scientists can use TrustyAI in OpenShift AI to monitor their data science models.

Data scientists can monitor their data science models in OpenShift AI for the following metrics:

Bias

Check for unfair patterns or biases in data and model predictions to ensure your model's decisions are unbiased.

Data drift

Detect changes in input data distributions over time by comparing the latest real-world data to the original training data. Comparing the data identifies shifts or deviations that could impact model performance, ensuring that the model remains accurate and reliable.

CHAPTER 2. CONFIGURING TRUSTYAI

To configure model monitoring with TrustyAI for data scientists to use in OpenShift AI, a cluster administrator does the following tasks:

- Configure monitoring for the model serving platform
- Enable the TrustyAI component in the Red Hat OpenShift AI Operator
- Configure TrustyAI to use a database, if you want to use your database instead of a PVC for storage with TrustyAI
- Install the TrustyAI service on each data science project that contains models that the data scientists want to monitor

2.1. CONFIGURING MONITORING FOR YOUR MODEL SERVING PLATFORM

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. Each model is deployed from its own model server. Use the single model serving platform in situations where you need 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 useful on OpenShift AI clusters that have finite compute resources or pods.

The process for configuring monitoring for either a single (KServe) or a multi-model (ModelMesh) serving platform is the same.

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
```

2.2. ENABLING THE TRUSTYAI COMPONENT

To allow your data scientists to use model monitoring with TrustyAI, you must enable the TrustyAI component in OpenShift AI.

Prerequisites

- You have cluster administrator privileges for your OpenShift cluster.
- You have access to the data science cluster.
- You have installed Red Hat OpenShift AI.

Procedure

1. In the OpenShift console, click **Operators → Installed Operators**.
2. Search for the **Red Hat OpenShift AI** Operator, and then click the Operator name to open the Operator details page.
3. Click the **Data Science Cluster** tab.
4. Click the default instance name (for example, **default-dsc**) to open the instance details page.
5. Click the **YAML** tab to show the instance specifications.
6. In the **spec:components** section, set the **managementState** field for the **trustyai** component to **Managed**:

```
trustyai:
  managementState: Managed
```

7. Click **Save**.

Verification

Check the status of the **trustyai-service-operator** pod:

1. In the OpenShift console, from the **Project** list, select **redhat-ods-applications**.
2. Click **Workloads → Deployments**.
3. Search for the **trustyai-service-operator-contoller-manager** deployment. Check the status:
   a. Click the deployment name to open the deployment details page.
   b. Click the **Pods** tab.
   c. View the pod status.
   When the status of the **trustyai-service-operator-controller-manager-*<pod-id>*** pod is **Running**, the pod is ready to use.

2.3. CONFIGURING TRUSTYAI WITH A DATABASE

If you have a relational database in your OpenShift cluster such as MySQL or MariaDB, you can configure TrustyAI to use your database instead of a persistent volume claim (PVC). Using a database instead of a PVC for storage can improve scalability, performance, and data management in TrustyAI. Provide TrustyAI with a database configuration secret before deployment. You can create a secret or specify the name of an existing Kubernetes secret within your project.

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 enabled the TrustyAI component, as described in Enabling the TrustyAI component.

- The data scientist has created a data science project, as described in Creating a data science project, that contains the models that the data scientist wants to monitor.
- If you are configuring the TrustyAI service with an external MySQL database, your database must already be in your cluster and use at least MySQL version 5.x. However, Red Hat recommends that you use MySQL version 8.x.
- If you are configuring the TrustyAI service with a MariaDB database, your database must already be in your cluster and use MariaDB version 10.3 or later. However, Red Hat recommends that you use at least MariaDB version 10.5.

NOTE

The transport security layer (TLS) protocol does not work with the MariaDB operator 0.29 or later versions.

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. Optional: If you want to use a TLS connection between TrustyAI and the database, create a TrustyAI service database TLS secret that uses the same certificates that you want to use for the database.
   a. Create a YAML file to contain your TLS secret and add the following code:

```
apiVersion: v1
kind: Secret
metadata:
  name: <service_name>-db-tls
type: kubernetes.io/tls
data:
  tls.crt: |
    <TLS CERTIFICATE>

  tls.key: |
    <TLS KEY>
```

   b. Save the file with the file name ***<service_name>*-db-tls.yaml**. For example, if your service name is **trustyai-service**, save the file as **trustyai-service-db-tls.yaml**.
   c. Apply the YAML file in the data science project that contains the models that the data scientist wants to monitor:

```
$ oc apply -f <service_name>-db-tls.yaml -n <project_name>
```

3. Create a secret (or specify an existing one) that has your database credentials.
   a. Create a YAML file to contain your secret and add the following code:

```
apiVersion: v1
kind: Secret
```

```
metadata:
  name: db-credentials
type: Opaque
stringData:
  databaseKind: <mariadb> (1)
  databaseUsername: <TrustyAI_username> (2)
  databasePassword: <TrustyAI_password> (3)
  databaseService: mariadb-service (4)
  databasePort: 3306 (5)
  databaseGeneration: update (6)
  databaseName: trustyai_service (7)
```

1. The only currently supported **databaseKind** value is **mariadb**.
2. The username you want TrustyAI to use when interfacing with the database.
3. The password that TrustyAI must use when connecting to the database.
4. The Kubernetes (K8s) service that TrustyAI must use when connecting to the database (the default **mariadb**) .
5. The port that TrustyAI must use when connecting to the database (default is 3306).
6. The database schema generation strategy to be used by TrustyAI. It is the setting for the **quarkus.hibernate-orm.database.generation** argument, which determines how TrustyAI interacts with the database on its initial connection. Set to **none**, **create**, **drop-and-create**, **drop**, **update**, or **validate**.
7. The name of the individual database within the database service that the username and password authenticate to, as well as the specific database name that TrustyAI should read and write to on the database server.

b. Save the file with the file name **db-credentials.yaml**. You will need this name later when you install or change the TrustyAI service.

c. Apply the YAML file in the data science project that contains the models that the data scientist wants to monitor:

```
$ oc apply -f db-credentials.yaml -n <project_name>
```

4. If you are installing TrustyAI for the first time on a project, continue to Installing the TrustyAI service for a project.
   If you already installed TrustyAI on a project, you can migrate the existing TrustyAI service from using a PVC to using a database.

a. Create a YAML file to update the TrustyAI service custom resource (CR) and add the following code:

```
apiVersion: trustyai.opendatahub.io/v1alpha1
kind: TrustyAIService
metadata:
  annotations:
    trustyai.opendatahub.io/db-migration: "true" (1)
  name: trustyai-service (2)
```

```
spec:
  storage:
    format: "DATABASE" 3
    folder: "/inputs" 4
      size: "1Gi" 5
    databaseConfigurations: <database_secret_credentials> 6
  data:
    filename: "data.csv" 7
  metrics:
    schedule: "5s" 8
```

1. Set to **true** to prompt the migration from PVC to database storage.
2. The name of the TrustyAI service instance.
3. The storage format for the data. Set this field to **DATABASE**.
4. The location within the PVC where you were storing the data. This must match the value specified in the existing CR.
5. The size of the data to request.
6. The name of the secret with your database credentials that you created in an earlier step. For example, **db-credentials**.
7. The suffix for the existing stored data files. This must match the value specified in the existing CR.
8. The interval at which to calculate the metrics. The default is **5s**. The duration is specified with the ISO-8601 format. For example, **5s** for 5 seconds, **5m** for 5 minutes, and **5h** for 5 hours.

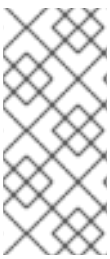
b. Save the file. For example, **trustyai_crd.yaml**.

c. Apply the new TrustyAI service CR to the data science project that contains the models that the data scientist wants to monitor:

```
$ oc apply -f trustyai_crd.yaml -n <project_name>
```

2.4. INSTALLING THE TRUSTYAI SERVICE FOR A PROJECT

Install the TrustyAI service on a data science project to provide access to its features for all models deployed within that project. An instance of the TrustyAI service is required for each data science project, or namespace, that contains models that the data scientists want to monitor.



NOTE

Install only one instance of the TrustyAI service in a project. Multiple instances in the same project can result in unexpected behavior.

Installing TrustyAI into a namespace where non-OVMS models are deployed can cause errors in the TrustyAI service.

2.4.1. Installing the TrustyAI service by using the CLI

You can use the OpenShift command-line interface (CLI) to install an instance of the TrustyAI service.

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 configured monitoring for the model serving platform, as described in Configuring monitoring for the multi-model serving platform.
- You have enabled the TrustyAI component, as described in Enabling the TrustyAI component.
- If you are using TrustyAI with a database instead of PVC, you have configured TrustyAI to use the database, as described in Configuring TrustyAI with a database.
- The data scientist has created a data science project, as described in Creating a data science project, that contains the models that the data scientist wants to monitor.

Procedure

1. Open a new terminal window.
2. Follow these steps to log in to your OpenShift cluster as a cluster administrator:
   a. In the OpenShift web console, click your user name and select **Copy login command**.
   b. After you have logged in, click **Display token**.
   c. Copy the **Log in with this token** command and paste it in the OpenShift command-line interface (CLI).

   ```
   $ oc login --token=<token> --server=<openshift_cluster_url>
   ```

3. Navigate to the data science project that contains the models that the data scientist wants to monitor.

   ```
   oc project <project_name>
   ```

   For example:

   ```
   oc project my-project
   ```

4. Create a **TrustyAIService** custom resource (CR) file, for example **trustyai_crd.yaml**:

   Example CR file for TrustyAI using a database

   ```
   apiVersion: trustyai.opendatahub.io/v1alpha1
   kind: TrustyAIService
   metadata:
     name: trustyai-service 1
   spec:
   ```

```
  storage:
    format: "DATABASE" 2
    size: "1Gi" 3
    databaseConfigurations: <database_secret_credentials> 4
  metrics:
    schedule: "5s" 5
```

1. The name of the TrustyAI service instance.
2. The storage format for the data, either **DATABASE** or **PVC** (persistent volume claim). Red Hat recommends that you use a database setup for better scalability, performance, and data management in TrustyAI.
3. The size of the data to request.
4. The name of the secret with your database credentials that you created in Configuring TrustyAI with a database. For example, **db-credentials**.
5. The interval at which to calculate the metrics. The default is **5s**. The duration is specified with the ISO-8601 format. For example, **5s** for 5 seconds, **5m** for 5 minutes, and **5h** for 5 hours.

Example CR file for TrustyAI using a PVC

```
apiVersion: trustyai.opendatahub.io/v1alpha1
kind: TrustyAIService
metadata:
  name: trustyai-service 1
spec:
  storage:
    format: "PVC" 2
    folder: "/inputs" 3
    size: "1Gi" 4
  data:
    filename: "data.csv" 5
    format: "CSV" 6
  metrics:
    schedule: "5s" 7
    batchSize: 5000 8
```

1. The name of the TrustyAI service instance.
2. The storage format for the data, either **DATABASE** or **PVC** (persistent volume claim).
3. The location within the PVC where you want to store the data.
4. The size of the PVC to request.
5. The suffix for the stored data files.
6. The format of the data. Currently, only comma-separated value (CSV) format is supported.

7 The interval at which to calculate the metrics. The default is **5s**. The duration is specified with the ISO-8601 format. For example, **5s** for 5 seconds, **5m** for 5 minutes, and **5h** for 5

8 (Optional) The observation's historical window size to use for metrics calculation. The default is **5000**, which means that the metrics are calculated using the 5,000 latest inferences.

5. Add the TrustyAI service's CR to your project:

```
oc apply -f trustyai_crd.yaml
```

This command returns output similar to the following:

```
trusty-service created
```

Verification

Verify that you installed the TrustyAI service:

```
oc get pods | grep trustyai
```

You should see a response similar to the following:

```
trustyai-service-5d45b5884f-96h5z          1/1     Running
```

CHAPTER 3. SETTING UP TRUSTYAI FOR YOUR PROJECT

To set up model monitoring with TrustyAI for a data science project, a data scientist does the following tasks:

- Authenticate the TrustyAI service
- Send training data to TrustyAI for bias or data drift monitoring
- Label your data fields (optional)

After setting up, a data scientist can create and view bias and data drift metrics for deployed models.

3.1. AUTHENTICATING THE TRUSTYAI SERVICE

To access TrustyAI service external endpoints, you must provide OAuth proxy (oauth-proxy) authentication. You must obtain a user token, or a token from a service account with sufficient privileges, and then pass the token to the TrustyAI service when using **curl** commands.

Prerequisites

- You installed the OpenShift command line interface (**oc**) as described in Installing the OpenShift CLI.
- Your OpenShift cluster administrator added you as a user to the OpenShift cluster and has installed the TrustyAI service for the data science project that contains the deployed models.

Procedure

1. Open a new terminal window.
2. Follow these steps to log in to your OpenShift cluster:
   a. In the upper-right corner of the OpenShift web console, click your user name and select **Copy login command**.
   b. After you have logged in, click **Display token**.
   c. Copy the **Log in with this token** command and paste it in the OpenShift command-line interface (CLI).

```
$ oc login --token=<token> --server=<openshift_cluster_url>
```

3. Enter the following command to set a user token variable on OpenShift:

```
export TOKEN=$(oc whoami -t)
```

Verification

- Enter the following command to check the user token variable:

```
echo $TOKEN
```

Next step

When running **curl** commands, pass the token to the TrustyAI service using the Authorization header. For example:

```
curl -H "Authorization: Bearer $TOKEN" $TRUSTY_ROUTE
```

3.2. SENDING TRAINING DATA TO TRUSTYAI

To use TrustyAI for bias monitoring or data drift detection, you must send training data for your model to TrustyAI.

Prerequisites

- Your OpenShift cluster administrator added you as a user to the OpenShift cluster and has installed the TrustyAI service for the data science project that contains the deployed models.
- You authenticated the TrustyAI service as described in Authenticating the TrustyAI service.
- Your deployed model is registered with TrustyAI.
  Verify that the TrustyAI service has registered your deployed model, as follows:
  1. In the OpenShift web console, navigate to **Workloads → Pods**.
  2. From the project list, select the project that contains your deployed model.
  3. Select the pod for your serving platform (for example, **modelmesh-serving-ovms-1.x-xxxxx**).
  4. On the **Environment** tab, verify that the **MM_PAYLOAD_PROCESSORS** environment variable is set.

Procedure

1. Set the **TRUSTY_ROUTE** variable to the external route for the TrustyAI service pod.

   ```
   TRUSTY_ROUTE=https://$(oc get route/trustyai-service --template={{.spec.host}})
   ```

2. Get the inference endpoints for the deployed model, as described in Accessing the inference endpoint for a deployed model.
3. Send data to this endpoint. For more information, see the KServe v2 Inference Protocol documentation.

Verification

Follow these steps to view cluster metrics and verify that TrustyAI is receiving data.

1. Log in to the OpenShift web console.
2. Switch to the **Developer** perspective.
3. In the left menu, click **Observe**.
4. On the **Metrics** page, click the **Select query** list and then select **Custom query**.

5. In the **Expression** field, enter **trustyai_model_observations_total** and press Enter. Your model should be listed and reporting observed inferences.

6. Optional: Select a time range from the list above the graph. For example, select **5m**.

3.3. LABELING DATA FIELDS

After you send model training data to TrustyAI, you might want to apply a set of name mappings to your inputs and outputs so that the field names are meaningful and easier to work with.

Prerequisites

- Your OpenShift cluster administrator added you as a user to the OpenShift cluster and has installed the TrustyAI service for the data science project that contains the deployed models.
- You sent training data to TrustyAI as described in Sending training data to TrustyAI.

Procedure

1. Open a new terminal window.
2. Follow these steps to log in to your OpenShift cluster:
   a. In the upper-right corner of the OpenShift web console, click your user name and select **Copy login command**.
   b. After you have logged in, click **Display token**.
   c. Copy the **Log in with this token** command and paste it in the OpenShift command-line interface (CLI).

   ```
   $ oc login --token=<token> --server=<openshift_cluster_url>
   ```

3. In the OpenShift CLI, get the route to the TrustyAI service:

   ```
   TRUSTY_ROUTE=https://$(oc get route/trustyai-service --template={{.spec.host}})
   ```

4. To examine TrustyAI's model metadata, query the **/info** endpoint:

   ```
   curl -H "Authorization: Bearer $TOKEN" $TRUSTY_ROUTE/info | jq ".[0].data"
   ```

   This outputs a JSON file containing the following information for each model:

   - The names, data types, and positions of input fields and output fields.
   - The observed field values.
   - The total number of input-output pairs observed.

5. Use **POST /info/names** to apply name mappings to the fields, similar to the following example. Change the **model-name**, **original-name**, and **Prediction** values to those used in your model. Change the **New name** values to the labels that you want to use.

   ```
   curl -sk -H "Authorization: Bearer $TOKEN" -X POST --location
   $TRUSTY_ROUTE/info/names \
   ```

```
-H "Content-Type: application/json"   \
-d "{
  \"modelId\": \"model-name\",
  \"inputMapping\":
    {
      \"original-name-0\": \"New name 0\",
      \"original-name-1\": \"New name 1\",
      \"original-name-2\": \"New name 2\",
      \"original-name-3\": \"New name 3\",
    },
  \"outputMapping\": {
    \"predict-0\": \"Prediction 0\"
  }
}"
```

For another example, see https://github.com/trustyai-explainability/odh-trustyai-demos/blob/main/2-BiasMonitoring/kserve-demo/scripts/apply_name_mapping.sh.

Verification

A "Feature and output name mapping successfully applied" message is displayed.

CHAPTER 4. MONITORING MODEL BIAS

As a data scientist, you might want to monitor your machine learning models for bias. This means monitoring for algorithmic deficiencies that might skew the outcomes or decisions that the model produces. Importantly, this type of monitoring helps you to ensure that the model is not biased against particular protected groups or features.

Red Hat OpenShift AI provides a set of metrics that help you to monitor your models for bias. You can use the OpenShift AI interface to choose an available metric and then configure model-specific details such as a protected attribute, the privileged and unprivileged groups, the outcome you want to monitor, and a threshold for bias. You then see a chart of the calculated values for a specified number of model inferences.

For more information about the specific bias metrics, see Supported bias metrics.

4.1. CREATING A BIAS METRIC

To monitor a deployed model for bias, you must first create bias metrics. When you create a bias metric, you specify details relevant to your model such as a protected attribute, privileged and unprivileged groups, a model outcome and a value that you want to monitor, and the acceptable threshold for bias.

For information about the specific bias metrics, see Supported bias metrics.

For the complete list of TrustyAI metrics, see TrustyAI service API.

You can create a bias metric for a model by using the OpenShift AI dashboard or by using the OpenShift command-line interface (CLI).

4.1.1. Creating a bias metric by using the dashboard

You can use the OpenShift AI dashboard to create a bias metric for a model.

Prerequisites

- You are familiar with the bias metrics that OpenShift AI supports and how to interpret them.
- You are familiar with the specific data set schema and understand the names and meanings of the inputs and outputs.
- Your OpenShift cluster administrator added you as a user to the OpenShift cluster and has installed the TrustyAI service for the data science project that contains the deployed models.
- You set up TrustyAI for your data science project, as described in Setting up TrustyAI for your project.

Procedure

1. Optional: To set the **TRUSTY_ROUTE** variable, follow these steps.
   a. In a terminal window, log in to the OpenShift cluster where OpenShift AI is deployed.

   ```
   oc login
   ```

   b. Set the **TRUSTY_ROUTE** variable to the external route for the TrustyAI service pod.

```
TRUSTY_ROUTE=https://$(oc get route/trustyai-service --template={{.spec.host}})
```

2. In the left menu of the OpenShift AI dashboard, click **Model Serving**.
3. On the **Deployed models** page, select your project from the drop-down list.
4. Click the name of the model that you want to configure bias metrics for.
5. On the metrics page for the model, click the **Model bias** tab.
6. Click **Configure**.
7. In the **Configure bias metrics** dialog, complete the following steps to configure bias metrics:
   a. In the **Metric name** field, type a unique name for your bias metric. Note that you cannot change the name of this metric later.
   b. From the **Metric type** list, select one of the metrics types that are available in OpenShift AI.
   c. In the **Protected attribute** field, type the name of an attribute in your model that you want to monitor for bias.

      TIP

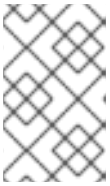
      You can use a **curl** command to query the metadata endpoint and view input attribute names and values. For example: **curl -H "Authorization: Bearer $TOKEN" $TRUSTY_ROUTE/info | jq ".[0].data.inputSchema"**

   d. In the **Privileged value** field, type the name of a privileged group for the protected attribute that you specified.
   e. In the **Unprivileged value** field, type the name of an unprivileged group for the protected attribute that you specified.
   f. In the **Output** field, type the name of the model outcome that you want to monitor for bias.

      TIP

      You can use a **curl** command to query the metadata endpoint and view output attribute names and values. For example: **curl -H "Authorization: Bearer $TOKEN" $TRUSTY_ROUTE/info | jq ".[0].data.outputSchema"**

   g. In the **Output value** field, type the value of the outcome that you want to monitor for bias.
   h. In the **Violation threshold** field, type the bias threshold for your selected metric type. This threshold value defines how far the specified metric can be from the fairness value for your metric, before the model is considered biased.
   i. In the **Metric batch size** field, type the number of model inferences that OpenShift AI includes each time it calculates the metric.
8. Ensure that the values you entered are correct.



NOTE

You cannot edit a model bias metric configuration after you create it. Instead, you can duplicate a metric and then edit (configure) it; however, the history of the original metric is not applied to the copy.

9. Click **Configure**.

Verification

- The **Bias metric configuration** page shows the bias metrics that you configured for your model.

Next step

To view metrics, on the **Bias metric configuration** page, click **View metrics** in the upper-right corner.

4.1.2. Creating a bias metric by using the CLI

You can use the OpenShift command-line interface (CLI) to create a bias metric for a model.

Prerequisites

- You are familiar with the bias metrics that OpenShift AI supports and how to interpret them.
- You are familiar with the specific data set schema and understand the names and meanings of the inputs and outputs.
- Your OpenShift cluster administrator added you as a user to the OpenShift cluster and has installed the TrustyAI service for the data science project that contains the deployed models.
- You set up TrustyAI for your data science project, as described in Setting up TrustyAI for your project.

Procedure

1. In a terminal window, log in to the OpenShift cluster where OpenShift AI is deployed.

```
oc login
```

2. Set the **TRUSTY_ROUTE** variable to the external route for the TrustyAI service pod.

```
TRUSTY_ROUTE=https://$(oc get route/trustyai-service --template={{.spec.host}})
```

3. Optionally, get the full list of TrustyAI service endpoints and payloads.

```
curl -H "Authorization: Bearer $TOKEN" --location $TRUSTY_ROUTE/q/openapi
```

4. Use **POST /metrics/group/fairness/spd/request** to schedule a recurring bias monitoring metric with the following syntax and payload structure:
Syntax:

```
curl -sk -H "Authorization: Bearer $TOKEN" -X POST --location
$TRUSTY_ROUTE/metrics/spd/request \
 --header 'Content-Type: application/json' \
```

```
--data <payload>
```

Payload structure:

modelId

The name of the model to query.

protectedAttribute

The name of the feature that distinguishes the groups that you are checking for fairness.

privilegedAttribute

The suspected favored (positively biased) class.

unprivilegedAttribute

The suspected unfavored (negatively biased) class.

outcomeName

The name of the output that provides the output you are examining for fairness.

favorableOutcome

The value of the **outcomeName** output that describes the favorable or desired model prediction.

batchSize

The number of previous inferences to include in the calculation.

For example:

```
curl -sk -H "Authorization: Bearer $TOKEN" -X POST --location
$TRUSTY_ROUTE/metrics/group/fairness/spd/ \
    --header 'Content-Type: application/json' \
    --data "{
                \"modelId\": \"demo-loan-nn-onnx-alpha\",
                \"protectedAttribute\": \"Is Male-Identifying?\",
                \"privilegedAttribute\": 1.0,
                \"unprivilegedAttribute\": 0.0,
                \"outcomeName\": \"Will Default?\",
                \"favorableOutcome\": 0,
                \"batchSize\": 5000
            }"
```

Verification

The bias metrics request should return output similar to the following:

```
{
  "timestamp":"2023-10-24T12:06:04.586+00:00",
  "type":"metric",
  "value":-0.0029676404469311524,
  "namedValues":null,
  "specificDefinition":"The SPD of -0.002968 indicates that the likelihood of Group:Is Male-
Identifying?=1.0 receiving Outcome:Will Default?=0 was -0.296764 percentage points lower than that
of Group:Is Male-Identifying?=0.0.",
  "name":"SPD",
  "id":"d2707d5b-cae9-41aa-bcd3-d950176cbbaf",
  "thresholds":{"lowerBound":-0.1,"upperBound":0.1,"outsideBounds":false}
}
```

The **specificDefinition** field helps you understand the real-world interpretation of these metric values. For this example, the model is fair over the **Is Male-Identifying?** field, with the rate of positive outcome only differing by about -0.3%.

4.1.3. Duplicating a bias metric

If you want to edit an existing metric, you can duplicate (copy) it in the OpenShift AI interface and then edit the values in the copy. However, note that the history of the original metric is not applied to the copy.

Prerequisites

- You are familiar with the bias metrics that OpenShift AI supports and how to interpret them.
- You are familiar with the specific data set schema and understand the names and meanings of the inputs and outputs.
- There is an existing bias metric that you want to duplicate.

Procedure

1. In the left menu of the OpenShift AI dashboard, click **Model Serving**.
2. On the **Deployed models** page, click the name of the model with the bias metric that you want to duplicate.
3. On the metrics page for the model, click the **Model bias** tab.
4. Click **Configure**.
5. On the **Bias metric configuration** page, click the action menu (⋮) next to the metric that you want to copy and then click **Duplicate**.
6. In the **Configure bias metric** dialog, follow these steps:
    a. In the **Metric name** field, type a unique name for your bias metric. Note that you cannot change the name of this metric later.
    b. Change the values of the fields as needed. For a description of these fields, see Creating a bias metric by using the dashboard.
7. Ensure that the values you entered are correct, and then click **Configure**.

Verification

- The **Bias metric configuration** page shows the bias metrics that you configured for your model.

Next step

To view metrics, on the **Bias metric configuration** page, click **View metrics** in the upper-right corner.

4.2. DELETING A BIAS METRIC

You can delete a bias metric for a model by using the OpenShift AI dashboard or by using the OpenShift command-line interface (CLI).

4.2.1. Deleting a bias metric by using the dashboard

You can use the OpenShift AI dashboard to delete a bias metric for a model.

Prerequisites

- You have logged in to Red Hat OpenShift AI.
- There is an existing bias metric that you want to delete.

Procedure

1. In the left menu of the OpenShift AI dashboard, click **Model Serving**.
2. On the **Deployed models** page, click the name of the model with the bias metric that you want to delete.
3. On the metrics page for the model, click the **Model bias** tab.
4. Click **Configure**.
5. Click the action menu (⋮) next to the metric that you want to delete and then click **Delete**.
6. In the **Delete bias metric** dialog, type the metric name to confirm the deletion.

NOTE

You cannot undo deleting a bias metric.

7. Click **Delete bias metric**

Verification

- The **Bias metric configuration** page does not show the bias metric that you deleted.

4.2.2. Deleting a bias metric by using the CLI

You can use the OpenShift command-line interface (CLI) to delete a bias metric for a model.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have a user token for authentication as described in Authenticating the TrustyAI service.
- There is an existing bias metric that you want to delete.

Procedure

1. Open a new terminal window.
2. Follow these steps to log in to your OpenShift cluster:
   a. In the upper-right corner of the OpenShift web console, click your user name and select **Copy login command**.

b. After you have logged in, click **Display token**.

c. Copy the **Log in with this token** command and paste it in the OpenShift command-line interface (CLI).

```
$ oc login --token=<token> --server=<openshift_cluster_url>
```

3. In the OpenShift CLI, get the route to the TrustyAI service:

```
TRUSTY_ROUTE=https://$(oc get route/trustyai-service --template={{.spec.host}})
```

4. Optional: To list all currently active requests for a metric, use **GET /metrics/{{metric}}/requests**. For example, to list all currently scheduled SPD metrics, type:

```
curl -H "Authorization: Bearer $TOKEN" -X GET --location
"$TRUSTY_ROUTE/metrics/spd/requests"
```

Alternatively, to list all currently scheduled metric requests, use **GET /metrics/all/requests**.

```
curl -H "Authorization: Bearer $TOKEN" -X GET --location
"$TRUSTY_ROUTE/metrics/all/requests"
```

5. To delete a metric, send an HTTP **DELETE** request to the **/metrics/$METRIC/request** endpoint to stop the periodic calculation, including the id of periodic task that you want to cancel in the payload. For example:

```
curl -H "Authorization: Bearer $TOKEN" -X DELETE --location
"$TRUSTY_ROUTE/metrics/spd/request" \
    -H "Content-Type: application/json" \
    -d "{
          \"requestId\": \"3281c891-e2a5-4eb3-b05d-7f3831acbb56\"
        }"
```

Verification

Use **GET /metrics/{{metric}}/requests** to list all currently active requests for the metric and verify the metric that you deleted is not shown. For example:

```
curl -H "Authorization: Bearer $TOKEN" -X GET --location "$TRUSTY_ROUTE/metrics/spd/requests"
```

4.3. VIEWING BIAS METRICS FOR A MODEL

After you create bias monitoring metrics, you can use the OpenShift AI dashboard to view and update the metrics that you configured.

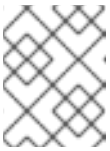
Prerequisite

- You configured bias metrics for your model as described in Creating a bias metric.

Procedure

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

2. On the **Deployed models** page, click the name of a model that you want to view bias metrics for.
3. On the metrics page for the model, click the **Model bias** tab.
4. To update the metrics shown on the page, follow these steps:
    a. In the **Metrics to display** section, use the **Select a metric** list to select a metric to show on the page.



NOTE

Each time you select a metric to show on the page, an additional **Select a metric** list appears. This enables you to show multiple metrics on the page.

    b. From the **Time range** list in the upper-right corner, select a value.
    c. From the **Refresh interval** list in the upper-right corner, select a value.
    The metrics page shows the metrics that you selected.
5. Optional: To remove one or more metrics from the page, in the **Metrics to display** section, perform one of the following actions:
    - To remove an individual metric, click the cancel icon (✖) next to the metric name.
    - To remove all metrics, click the cancel icon (✖) in the **Select a metric** list.
6. Optional: To return to configuring bias metrics for the model, on the metrics page, click **Configure** in the upper-right corner.

Verification

- The metrics page shows the metrics selections that you made.

4.4. SUPPORTED BIAS METRICS

Red Hat OpenShift AI supports the following bias metrics:

Statistical Parity Difference

Statistical Parity Difference (SPD) is the difference in the probability of a favorable outcome prediction between unprivileged and privileged groups. The formal definition of SPD is the following:

$$SPD = p(\hat{y} = 1 \mid D_u) - p(\hat{y} = 1 \mid D_p)$$

- $\hat{y} = 1$ is the favorable outcome.
- D_u and D_p are the unprivileged and privileged group data.

You can interpret SPD values as follows:

- A value of **0** means that the model is behaving fairly for a selected attribute (for example, race, gender).
- A value in the range **-0.1** to **0.1** means that the model is reasonably fair for a selected attribute. Instead, you can attribute the difference in probability to other factors, such as the sample size.

- A value outside the range **-0.1** to **0.1** indicates that the model is unfair for a selected attribute.
- A negative value indicates that the model has bias against the unprivileged group.
- A positive value indicates that the model has bias against the privileged group.

Disparate Impact Ratio

Disparate Impact Ratio (DIR) is the ratio of the probability of a favorable outcome prediction for unprivileged groups to that of privileged groups. The formal definition of DIR is the following:

$$DIR = \frac{p(\hat{y} = 1 \mid D_u)}{p(\hat{y} = 1 \mid D_p)}$$

- $\hat{y} = 1$ is the favorable outcome.
- D_u and D_p are the unprivileged and privileged group data.

The threshold to identify bias depends on your own criteria and specific use case.

For example, if your threshold for identifying bias is represented by a DIR value below **0.8** or above **1.2**, you can interpret the DIR values as follows:

- A value of **1** means that the model is fair for a selected attribute.
- A value of between **0.8** and **1.2** means that the model is reasonably fair for a selected attribute.
- A value below **0.8** or above **1.2** indicates bias.

CHAPTER 5. MONITORING DATA DRIFT

As a data scientist, you might want to monitor your deployed models for data drift. Data drift refers to changes in the distribution or properties of incoming data that differ significantly from the data on which the model was originally trained. Detecting data drift helps ensure that your models continue to perform as expected and that they remain accurate and reliable.

You can use data drift monitoring metrics from TrustyAI in Red Hat OpenShift AI to provide a quantitative measure of the alignment between the training data and the inference data.

For information about the specific data drift metrics, see Supported drift metrics.

5.1. CREATING A DRIFT METRIC

To monitor a deployed model for data drift, you must first create drift metrics.

For information about the specific data drift metrics, see Supported drift metrics.

For the complete list of TrustyAI metrics, see TrustyAI service API.

5.1.1. Creating a drift metric by using the CLI

You can use the OpenShift command-line interface (CLI) to create a data drift metric for a model.

Prerequisites

- You are familiar with the specific data set schema and understand the relevant inputs and outputs.
- Your OpenShift cluster administrator added you as a user to the OpenShift cluster and has installed the TrustyAI service for the data science project that contains the deployed models.
- You set up TrustyAI for your data science project, as described in Setting up TrustyAI for your project.

Procedure

1. Open a new terminal window.
2. Follow these steps to log in to your OpenShift cluster:
   a. In the upper-right corner of the OpenShift web console, click your user name and select **Copy login command**.
   b. After you have logged in, click **Display token**.
   c. Copy the **Log in with this token** command and paste it in the OpenShift command-line interface (CLI).

   ```
   $ oc login --token=<token> --server=<openshift_cluster_url>
   ```

3. Set the **TRUSTY_ROUTE** variable to the external route for the TrustyAI service pod.

   ```
   TRUSTY_ROUTE=https://$(oc get route/trustyai-service --template={{.spec.host}})
   ```

4. Optionally, get the full list of TrustyAI service endpoints and payloads.

```
curl -H "Authorization: Bearer $TOKEN" --location $TRUSTY_ROUTE/q/openapi
```

5. Use **POST /metrics/drift/meanshift/request** to schedule a recurring drift monitoring metric with the following syntax and payload structure:
Syntax:

```
curl -k -H "Authorization: Bearer $TOKEN" -X POST --location
$TRUSTY_ROUTE/metrics/drift/meanshift/request \
 --header 'Content-Type: application/json' \
 --data <payload>
```

Payload structure:

modelId

The name of the model to monitor.

referenceTag

The data to use as the reference distribution.

For example:

```
curl -k -H "Authorization: Bearer $TOKEN" -X POST --location
$TRUSTY_ROUTE/metrics/drift/meanshift/request \
    --header 'Content-Type: application/json' \
    --data "{
                \"modelId\": \"gaussian-credit-model\",
                \"referenceTag\": \"TRAINING\"
            }"
```

5.2. DELETING A DRIFT METRIC BY USING THE CLI

You can use the OpenShift command-line interface (CLI) to delete a drift metric for a model.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have a user token for authentication as described in Authenticating the TrustyAI service.
- There is an existing drift metric that you want to delete.

Procedure

1. Open a new terminal window.
2. Follow these steps to log in to your OpenShift cluster:
   a. In the OpenShift web console, click your user name and select **Copy login command**.
   b. After you have logged in, click **Display token**.

c. Copy the **Log in with this token** command and paste it in the OpenShift command-line interface (CLI).

```
$ oc login --token=<token> --server=<openshift_cluster_url>
```

3. In the OpenShift CLI, get the route to the TrustyAI service:

```
TRUSTY_ROUTE=https://$(oc get route/trustyai-service --template={{.spec.host}})
```

4. Optional: To list all currently active requests for a metric, use **GET /metrics/{{metric}}/requests**. For example, to list all currently scheduled MeanShift metrics, type:

```
curl -k -H "Authorization: Bearer $TOKEN" -X GET --location
"$TRUSTY_ROUTE/metrics/drift/meanshift/requests"
```

Alternatively, to list all currently scheduled metric requests, use **GET /metrics/all/requests**.

```
curl -H "Authorization: Bearer $TOKEN" -X GET --location
"$TRUSTY_ROUTE/metrics/all/requests"
```

5. To delete a metric, send an HTTP **DELETE** request to the **/metrics/$METRIC/request** endpoint to stop the periodic calculation, including the id of periodic task that you want to cancel in the payload. For example:

```
curl -k -H "Authorization: Bearer $TOKEN" -X DELETE --location
"$TRUSTY_ROUTE/metrics/drift/meanshift/request" \
    -H "Content-Type: application/json" \
    -d "{
          \"requestId\": \"$id\"
        }"
```

Verification

Use **GET /metrics/{{metric}}/requests** to list all currently active requests for the metric and verify the metric that you deleted is not shown. For example:

```
curl -H "Authorization: Bearer $TOKEN" -X GET --location
"$TRUSTY_ROUTE/metrics/drift/meanshift/requests"
```

5.3. VIEWING DATA DRIFT METRICS FOR A MODEL

After you create data drift monitoring metrics, use the OpenShift web console to view and update the metrics that you configured.

Prerequisites

- You have been assigned the **monitoring-rules-view** role. For more information, see Granting users permission to configure monitoring for user-defined projects.
- You are familiar with how to monitor project metrics in the OpenShift web console. For more information, see Monitoring your project metrics.

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 your project metrics, use the web console to run queries for **trustyai_*** metrics.

5.4. SUPPORTED DRIFT METRICS

Red Hat OpenShift AI supports the following data drift metrics:

MeanShift

The MeanShift metric calculates the per-column probability that the data values in a test data set are from the same distribution as those in a training data set (assuming that the values are normally distributed). This metric measures the difference in the means of specific features between the two datasets.
MeanShift is useful for identifying straightforward changes in data distributions, such as when the entire distribution has shifted to the left or right along the feature axis.

This metric returns the probability that the distribution seen in the "real world" data is derived from the same distribution as the reference data. The closer the value is to 0, the more likely there is to be significant drift.

FourierMMD

The FourierMMD metric provides the probability that the data values in a test data set have drifted from the training data set distribution, assuming that the computed Maximum Mean Discrepancy (MMD) values are normally distributed. This metric compares the empirical distributions of the data sets by using an MMD measure in the Fourier domain.
FourierMMD is useful for detecting subtle shifts in data distributions that might be overlooked by simpler statistical measures.

This metric returns the probability that the distribution seen in the "real world" data has drifted from the reference data. The closer the value is to 1, the more likely there is to be significant drift.

KSTest

The KSTest metric calculates two Kolmogorov-Smirnov tests for each column to determine whether the data sets are derived from the same distributions. This metric measures the maximum distance between the empirical cumulative distribution functions (CDFs) of the data sets, without assuming any specific underlying distribution.
KSTest is useful for detecting changes in distribution shape, location, and scale.

This metric returns the probability that the distribution seen in the "real world" data is derived from the same distribution as the reference data. The closer the value is to 0, the more likely there is to be significant drift.

ApproxKSTest

The ApproxKSTest metric performs an approximate Kolmogorov-Smirnov test, ensuring that the maximum error is **6*epsilon** compared to an exact KSTest.
ApproxKSTest is useful for detecting changes in distributions for large data sets where performing an exact KSTest might be computationally expensive.

This metric returns the probability that the distribution seen in the "real world" data is derived from the same distribution as the reference data. The closer the value is to 0, the more likely there is to be significant drift.

CHAPTER 6. EVALUATING LARGE LANGUAGE MODELS

A large language model (LLM) is a type of artificial intelligence (AI) program that is designed for natural language processing tasks, such as recognizing and generating text.

As a data scientist, you might want to monitor your large language models against a range of metrics, in order to ensure the accuracy and quality of its output. Features such as summarization, language toxicity, and question-answering accuracy can be assessed to inform and improve your model parameters.

Red Hat OpenShift AI now offers Language Model Evaluation as a Service (LM-Eval-aaS), in a feature called LM-Eval. LM-Eval provides a unified framework to test generative language models on a vast range of different evaluation tasks.

The following sections show you how to create an **LMEvalJob** custom resource (CR) which allows you to activate an evaluation job and generate an analysis of your model's ability.

6.1. SETTING UP LM-EVAL

LM-Eval is a service designed for evaluating large language models that has been integrated into the TrustyAI Operator.

The service is built on top of two open-source projects:

- LM Evaluation Harness, developed by EleutherAI, that provides a comprehensive framework for evaluating language models
- Unitxt, a tool that enhances the evaluation process with additional functionalities

The following information explains how to create an **LMEvalJob** custom resource (CR) to initiate an evaluation job and get the results.

Global settings for LM-Eval

Configurable global settings for LM-Eval services are stored in the TrustyAI operator global **ConfigMap**, named **trustyai-service-operator-config**. The global settings are located in the same namespace as the operator.

You can configure the following properties for LM-Eval:

Table 6.1. LM-Eval properties

Property	Default	Description
lmes-detect-device	**true/false**	Detect if there are GPUs available and assign a value for **--device argument** for LM Evaluation Harness. If GPUs are available, the value is **cuda**. If there are no GPUs available, the value is **cpu**.
lmes-pod-image	**quay.io/trustyai/ta-lmes-job:latest**	The image for the LM-Eval job. The image contains the Python packages for LM Evaluation Harness and Unitxt.

Property	Default	Description
lmes-driver-image	**quay.io/trustyai/ta-lmes-driver:latest**	The image for the LM-Eval driver. For detailed information about the driver, see the **cmd/lmes_driver** directory.
lmes-image-pull-policy	**Always**	The image-pulling policy when running the evaluation job.
lmes-default-batch-size	8	The default batch size when invoking the model inference API. Default batch size is only available for local models.
lmes-max-batch-size	24	The maximum batch size that users can specify in an evaluation job.
lmes-pod-checking-interval	10s	The interval to check the job pod for an evaluation job.
lmes-allow-online	true	Whether LMEval jobs can set the online mode to **on** to access artifacts (models, datasets, tokenizers) from the internet.
lmes-code-execution	true	Determines whether LMEval jobs can set the **trust remote code** mode to **on**.

After updating the settings in the **ConfigMap**, restart the operator to apply the new values.

IMPORTANT

The **allowOnline** setting is disabled by default at the operator level in Red Hat OpenShift AI, as using **allowOnline** gives the job permissions to automatically download artifacts from external sources.

Enabling allowOnline mode

To enable **allowOnline** mode, patch the TrustyAI operator **ConfigMap** with the following code:

```
kubectl patch configmap trustyai-service-operator-config -n redhat-ods-applications \
--type merge -p '{"data":{"lmes-allow-online":"true","lmes-allow-code-execution":"true"}}'
```

Then restart the TrustyAI operator with:

```
kubectl rollout restart deployment trustyai-service-operator-controller-manager -n redhat-ods-applications
```

6.2. LM-EVAL EVALUATION JOB

LM-Eval service defines a new Custom Resource Definition (CRD) called **LMEvalJob**. An **LMEvalJob** object represents an evaluation job. **LMEvalJob** objects are monitored by the TrustyAI Kubernetes operator.

To run an evaluation job, create an **LMEvalJob** object with the following information: **model**, **model arguments**, **task**, and **secret**.

After the **LMEvalJob** is created, the LM-Eval service runs the evaluation job. The status and results of the **LMEvalJob** object update when the information is available.

NOTE

Other TrustyAI features (such as bias and drift metrics) do not support non-tabular models (including LLMs). Deploying the **TrustyAIService** custom resource (CR) in a namespace that contains non-tabular models (such as the namespace where an evaluation job is being executed) can cause errors within the TrustyAI service.

Sample LMEvalJob object

The sample **LMEvalJob** object contains the following features:

- The **google/flan-t5-base** model from Hugging Face.
- The dataset from the **wnli** card, a subset of the GLUE (General Language Understanding Evaluation) benchmark evaluation framework from Hugging Face. For more information about the **wnli** Unitxt card, see the Unitxt website.
- The following default parameters for the **multi_class.relation** Unitxt task: **f1_micro**, **f1_macro**, and **accuracy**. This template can be found on the Unitxt website: click **Catalog**, then click **Tasks** and select **Classification** from the menu.

The following is an example of an **LMEvalJob** object:

```
apiVersion: trustyai.opendatahub.io/v1alpha1
kind: LMEvalJob
metadata:
  name: evaljob-sample
spec:
  model: hf
  modelArgs:
  - name: pretrained
    value: google/flan-t5-base
  taskList:
    taskRecipes:
    - card:
        name: "cards.wnli"
      template: "templates.classification.multi_class.relation.default"
  logSamples: true
```

After you apply the sample **LMEvalJob**, check its state by using the following command:

```
oc get lmevaljob evaljob-sample
```

Output similar to the following appears: NAME: **evaljob-sample** STATE: **Running**

Evaluation results are available when the state of the object changes to **Complete**. Both the model and dataset in this example are small. The evaluation job should finish within 10 minutes on a CPU-only node.

Use the following command to get the results:

```
oc get lmevaljobs.trustyai.opendatahub.io evaljob-sample \
  -o template --template={{.status.results}} | jq '.results'
```

The command returns results similar to the following example:

```
{
  "tr_0": {
    "alias": "tr_0",
    "f1_micro,none": 0.5633802816901409,
    "f1_micro_stderr,none": "N/A",
    "accuracy,none": 0.5633802816901409,
    "accuracy_stderr,none": "N/A",
    "f1_macro,none": 0.36036036036036034,
    "f1_macro_stderr,none": "N/A"
  }
}
```

Notes on the results

- The **f1_micro**, **f1_macro**, and **accuracy** scores are 0.56, 0.36, and 0.56.
- The full results are stored in the **.status.results** of the **LMEvalJob** object as a JSON document.
- The command above only retrieves the results field of the JSON document.

LMEvalJob properties

The following table lists each property in the **LMEvalJob** and its usage:

Table 6.2. LM-Eval properties

Parameter	Description

Parameter	Description
model	Specifies which model type or provider is evaluated. This field directly maps to the **--model** argument of the**lm-evaluation-harness**. Supported model types and providers include: • **hf**: HuggingFace models • **openai-completions**: OpenAI Completions API models • **openai-chat-completions**: OpenAI Chat Completions API models • **local-completions** and **local-chat-completions**: OpenAI API-compatible servers • **textsynth**: TextSynth APIs
modelArgs	A list of paired name and value arguments for the model type. Each model type or provider supports different arguments. You can find further details in the models section of the LM Evaluation Harness library on GitHub. • **hf** (HuggingFace) • **local-completions** (An OpenAI API-compatible server) • **local-chat-completions** (An OpenAI API-compatible server) • **openai-completions** (OpenAI Completions API models) • **openai-chat-completions** (ChatCompletions API models) • **textsynth** (TextSynth APIs)
taskList.taskNames	Specifies a list of tasks supported by **lm-evaluation-harness**.

Parameter	Description

taskList.taskRecipes	Specifies the task using the Unitxt recipe format:
	• **card**: Use the **name** to specify a Unitxt card or **custom** for a custom card.
	  ◦ **name**: Specifies a Unitxt card from the Unitxt catalog. Use the card ID as the value. For example, the ID of the Wnli card is **cards.wnli**.
	  ◦ **custom**: Defines and uses a custom card. The value is a JSON object that contains the custom dataset. For more information about creating a custom card, see the Unitxt documentation on their website. If the dataset used by the custom card requires an API key from an environment variable or a persistent volume, configure the necessary resources in the **pod** field.
	• **template**: Specifies a Unitxt template from the Unitxt catalog. Use the template ID as the value.
	• **task** (optional): Specifies a Unitxt task from the Unitxt catalog. Use the task ID as the value. A Unitxt card has a predefined task. Only specify a value for this if you want to run a different task.
	• **metrics** (optional): Specifies a Unitxt task from the Unitxt catalog. Use the metric ID as the value. A Unitxt task has a set of pre-defined metrics. Only specify a set of metrics if you need different metrics.
	• **format** (optional): Specifies a Unitxt format from the Unitxt catalog. Use the format ID as the value.
	• **loaderLimit** (optional): Specifies the maximum number of instances per stream to be returned from the loader. You can use this parameter to reduce loading time in large datasets.
	• **numDemos** (optional): Number of few-shot to be used.
	• **demosPoolSize** (optional): Size of the few-shot pool.

numFewShot	Sets the number of few-shot examples to place in context. If you are using a task from Unitxt, do not use this field. Use **numDemos** under the **taskRecipes** instead.
limit	Set a limit to run the tasks instead of running the entire dataset. Accepts either an integer or a float between 0.0 and 1.0.
genArgs	Maps to the **--gen_kwargs** parameter for the **lm-evaluation-harness**. For more information, see the LM Evaluation Harness documentation on GitHub.
logSamples	If this flag is passed, then the model outputs and the text fed into the model will be saved at per-document granularity.

Parameter	Description
batchSize	Specifies the batch size for the evaluation in integer format. The **auto:N** batch size is not used for API models, but numeric batch sizes are used for APIs.

pod	Specifies extra information for the **lm-eval** job pod:
	• **container**: Specifies additional container settings for the **lm-eval** container.
	  ◦ **env**: Specifies environment variables. This parameter uses the **EnvVar** data structure of Kubernetes.
	  ◦ **volumeMounts**: Mounts the volumes into the **lm-eval** container.
	  ◦ **resources**: Specifies the resources for the **lm-eval** container.
	• **volumes**: Specifies the volume information for the **lm-eval** and other containers. This parameter uses the **Volume** data structure of Kubernetes.
	• **sideCars**: A list of containers that run along with the **lm-eval** container. It uses the **Container** data structure of Kubernetes.

outputs	This parameter defines a custom output location to store the the evaluation results. Only Persistent Volume Claims (PVC) are supported.
outputs.pvcManaged	Creates an operator-managed PVC to store the job results. The PVC is named **<job-name>-pvc** and is owned by the **LMEvalJob**. After the job finishes, the PVC is still be available, but it is deleted with the **LMEvalJob**. Supports the following fields: • **size**: The PVC size, compatible with standard PVC syntax (for example, 5Gi)
outputs.pvcName	Binds an existing PVC to a job by specifying its name. The PVC must be created separately and must already exist when creating the job.
allowOnline	If this parameter is set to **true**, the LMEval job downloads artifacts as needed (for example, models, datasets or tokenizers). If set to **false**, artifacts are not downloaded and are pulled from local storage instead. This setting is disabled by default. If you want to enable **allowOnline** mode, you can patch the TrustyAI operator **ConfigMap**.
allowCodeExecution	If this parameter is set to **true**, the LMEval job executes the necessary code for preparing models or datasets. If set to **false** it does not execute downloaded code.
offline	Mount a PVC as the local storage for models and datasets.

6.3. LM-EVAL SCENARIOS

The following procedures outline example scenarios that can be useful for an ML-Eval setup.

6.3.1. Configuring the LM-Eval environment

If the **LMEvalJob** needs to access a model on HuggingFace with the access token, you can set up the **HF_TOKEN** as one of the environment variables for the **lm-eval** container.

Prerequisites

- You have logged in to Red Hat OpenShift AI.
- Your OpenShift cluster administrator has installed OpenShift AI and enabled the TrustyAI service for the data science project where the models are deployed.

Procedure

1. To start an evaluation job for a **huggingface** model, apply the following YAML file:

```
apiVersion: trustyai.opendatahub.io/v1alpha1
kind: LMEvalJob
metadata:
  name: evaljob-sample
spec:
  model: hf
  modelArgs:
  - name: pretrained
    value: huggingfacespace/model
  taskList:
    taskNames:
    - unfair_tos/
  logSamples: true
  pod:
    container:
      env:
      - name: HF_TOKEN
        value: "My HuggingFace token"
```

2. (Optional) You can also create a secret to store the token, then refer the key from the **secretKeyRef** object using the following reference syntax:

```
env:
  - name: HF_TOKEN
    valueFrom:
      secretKeyRef:
        name: my-secret
        key: hf-token
```

6.3.2. Using a custom Unitxt card

You can run evaluations using custom Unitxt cards. To do this, include the custom Unitxt card in JSON format within the **LMEvalJob** YAML.

Prerequisites

- You have logged in to Red Hat OpenShift AI.
- Your OpenShift cluster administrator has installed OpenShift AI and enabled the TrustyAI service for the data science project where the models are deployed.

Procedure

1. Pass a custom Unitxt Card in JSON format:

```
apiVersion: trustyai.opendatahub.io/v1alpha1
kind: LMEvalJob
metadata:
  name: evaljob-sample
spec:
  model: hf
  modelArgs:
  - name: pretrained
    value: google/flan-t5-base
  taskList:
    taskRecipes:
    - template: "templates.classification.multi_class.relation.default"
      card:
        custom: |
          {
            "__type__": "task_card",
            "loader": {
              "__type__": "load_hf",
              "path": "glue",
              "name": "wnli"
            },
            "preprocess_steps": [
              {
                "__type__": "split_random_mix",
                "mix": {
                  "train": "train[95%]",
                  "validation": "train[5%]",
                  "test": "validation"
                }
              },
              {
                "__type__": "rename",
                "field": "sentence1",
                "to_field": "text_a"
              },
              {
                "__type__": "rename",
                "field": "sentence2",
                "to_field": "text_b"
              },
              {
                "__type__": "map_instance_values",
                "mappers": {
                  "label": {
                    "0": "entailment",
```

```
                "1": "not entailment"
              }
            }
          },
          {
            "__type__": "set",
            "fields": {
              "classes": [
                "entailment",
                "not entailment"
              ]
            }
          },
          {
            "__type__": "set",
            "fields": {
              "type_of_relation": "entailment"
            }
          },
          {
            "__type__": "set",
            "fields": {
              "text_a_type": "premise"
            }
          },
          {
            "__type__": "set",
            "fields": {
              "text_b_type": "hypothesis"
            }
          }
        ],
        "task": "tasks.classification.multi_class.relation",
        "templates": "templates.classification.multi_class.relation.all"
      }
    logSamples: true
```

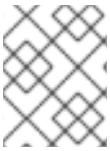
2. Inside the custom card specify the Hugging Face dataset loader:

```
"loader": {
            "__type__": "load_hf",
            "path": "glue",
            "name": "wnli"
          },
```

3. (Optional) You can use other Unitxt loaders (found on the Unitxt website) that contain the **volumes** and **volumeMounts** parameters to mount the dataset from persistent volumes. For example, if you use the **LoadCSV** Unitxt command, mount the files to the container and make the dataset accessible for the evaluation process.

6.3.3. Using PVCs as storage

To use a PVC as storage for the **LMEvalJob** results, you can use either managed PVCS or existing PVCs. Managed PVCs are managed by the TrustyAI operator. Existing PVCs are created by the end-user before the **LMEvalJob** is created.



NOTE

If both managed and existing PVCs are referenced in outputs, the TrustyAI operator defaults to the managed PVC.

Prerequisites

- You have logged in to Red Hat OpenShift AI.
- Your OpenShift cluster administrator has installed OpenShift AI and enabled the TrustyAI service for the data science project where the models are deployed.

6.3.3.1. Managed PVCs

To create a managed PVC, specify its size. The managed PVC is named **<job-name>-pvc** and is available after the job finishes. When the **LMEvalJob** is deleted, the managed PVC is also deleted.

Procedure

- Enter the following code:

```
apiVersion: trustyai.opendatahub.io/v1alpha1
kind: LMEvalJob
metadata:
  name: evaljob-sample
spec:
  # other fields omitted ...
  outputs:
    pvcManaged:
      size: 5Gi
```

Notes on the code

- **outputs** is the section for specifying custom storage locations
- **pvcManaged** will create an operator-managed PVC
- **size** (compatible with standard PVC syntax) is the only supported value

6.3.3.2. Existing PVCs

To use an existing PVC, pass its name as a reference. The PVC must exist when you create the **LMEvalJob**. The PVC is not managed by the TrustyAI operator, so it is available after deleting the **LMEvalJob**.

Procedure

1. Create a PVC. An example is the following:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: "my-pvc"
spec:
```

```
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

2. Reference the new PVC from the **LMEvalJob**.

```
apiVersion: trustyai.opendatahub.io/v1alpha1
kind: LMEvalJob
metadata:
  name: evaljob-sample
spec:
  # other fields omitted ...
  outputs:
    pvcName: "my-pvc"
```

6.3.4. Using an InferenceService

To run an evaluation job on an **InferenceService** which is already deployed and running in your namespace, define your LMEvalJob CR, then apply this CR into the same namespace as your model.

Prerequisites

- You have logged in to Red Hat OpenShift AI.
- Your OpenShift cluster administrator has installed OpenShift AI and enabled the TrustyAI service for the data science project where the models are deployed.
- You have a namespace that contains an InferenceService with a vLLM model. This example assumes that the vLLM model is already deployed in your cluster.

Procedure

1. Define your **LMEvalJob** CR:

```
  apiVersion: trustyai.opendatahub.io/v1alpha1
kind: LMEvalJob
metadata:
  name: evaljob
spec:
  model: local-completions
  taskList:
    taskNames:
      - mmlu
  logSamples: true
  batchSize: 1
  modelArgs:
    - name: model
      value: granite
    - name: base_url
      value: $ROUTE_TO_MODEL/v1/completions
    - name: num_concurrent
      value:  "1"
    - name: max_retries
```

```
      value:  "3"
    - name: tokenized_requests
      value: "False"
    - name: tokenizer
      value: ibm-granite/granite-7b-instruct
env:
  - name: OPENAI_TOKEN
    valueFrom:
        secretKeyRef:
          name: <secret-name>
          key: token
```

2. Apply this CR into the same namespace as your model.

Verification

A pod spins up in your model namespace called **evaljob**. In the pod terminal, you can see the output via **tail -f output/stderr.log**.

Notes on the code

- **base_url** should be set to the route/service URL of your model. Make sure to include the /**v1**/**completions** endpoint in the URL.
- **env.valueFrom.secretKeyRef.name** should point to a secret that contains a token that can authenticate to your model. **secretRef.name** should be the secret's name in the namespace, while **secretRef.key** should point at the token's key within the secret.
- **secretKeyRef.name** can equal the output of:

```
oc get secrets -o custom-columns=SECRET:.metadata.name --no-headers | grep user-one-token
```

- **secretKeyRef.key** is set to **token**