Oracle® Communications Billing and Revenue Management

AI Deployment and Integration Guide

Release 15.2
G49673-01
January 2026

ORACLE

Oracle Communications Billing and Revenue Management AI Deployment and Integration Guide, Release 15.2

G49673-01

Contents

About This Content

Describes how to integrate AI-based microservices with Oracle Monetization Suite.

Audience

This guide is intended for developers.

1

Getting Started with AI in Oracle Monetization Suite

Learn how to get started with artificial intelligence (AI) using the deployment assets provided for Oracle Monetization Suite. These assets include container images, Helm charts, configuration files, and other components that assist you in installing, deploying, and managing AI services on Kubernetes. You can use these resources in both Oracle Cloud Infrastructure (OCI) and other supported environments.

Topics in the document:

Overview of the Deployment Package

The deployment package provides all components required to deploy, manage, and monitor data, training, and prediction services for AI-based workflows on Kubernetes. The package includes:

- Ready-to-use Docker images and Helm charts for each service, allowing you to orchestrate containers in your Kubernetes environment. Update the **values.yaml** file as needed to reflect your specific configuration requirements.
- Dockerfiles and build files that enable you to apply minor updates to the base operating system or add additional dependencies.
- Sample configuration files, Python scripts, and shell scripts that serve as references for customizing your deployment framework.
- Templates for Kubernetes persistent storage, which you can modify to suit your application data and artifact storage needs.
- Sample Grafana dashboard JSON files for each service, designed to streamline performance monitoring and observability with Prometheus and Tempo.

Tasks for Preparing Your Cloud Native Environment

Prepare your system for the BRM cloud native deployment by performing the following high-level tasks:

1. Downloading the Helm charts for the cloud native deployment. See "Downloading Packages for the Cloud Native Helm Charts and Docker Files".

2. Downloading the cloud native images in one of these ways:
   a. From the Oracle Container Registry. To do so, see "Pulling Images from the Oracle Container Registry".
   b. From the Oracle Software Delivery website. To do so, see "Downloading Images from Oracle Software Delivery Website".

Downloading Packages for the Cloud Native Helm Charts and Docker Files

To download the packages for the cloud native helm charts and docker files, see "Downloading Packages for the BRM Cloud Native Helm Charts" in *BRM Cloud Native Deployment Guide*.

Table 1-1 lists the package name and archive files that need to be extracted for AI and ML services.

Table 1-1 Packages and Archive Files

Package Name	Archive Files
Oracle Communications Cloud Native ML Helm Charts 15.2.0	**ml-helmcharts.tgz** Includes: • **data-fetch-15.2.0.0.0.tgz** • **train-utility-15.2.0.0.0.tgz** • **recommendation-15.2.0.0.0.tgz**
Oracle Communications Cloud Native ML Docker files and Artifacts Bundle 15.2.0	**ml-dockerfiles-artifacts.tgz** Includes: • **ml-data-fetch-orchestrator-artifacts.zip** • **ml-data-fetch-processor-artifacts.zip** • **ml-train-utility-orchestrator-artifacts.zip** • **ml-train-utility-processor-artifacts.zip** • **ml-recommendation-orchestrator-artifacts.zip** • **ml-recommendation-predictor-artifacts.zip**

Pulling Images from the Oracle Container Registry

To pull images from the Oracle Container Registry, see "Pulling BRM Images from the Oracle Container Registry" in *BRM Cloud Native Deployment Guide*.

Table 1-2 lists the image names for the AI-ML cloud native components.

Table 1-2 Cloud Native Images

Component Name	Image Name
ML Data Fetch Orchestrator	**ml-data-fetch-orchestrator**
ML Data Fetch Processor	**ml-data-fetch-processor**
ML Train Utility Orchestrator	**ml-train-utility-orchestrator**
ML Train Utility Processor	**ml-train-utility-predictor**
ML Recommendation Orchestrator	**ml-recommendation-orchestrator**

Table 1-2 (Cont.) Cloud Native Images

Component Name	Image Name
ML Recommendation Predictor	**ml-recommendation-predictor**

Downloading Images from Oracle Software Delivery Website

To download images from Oracle Software Delivery Website, see "Downloading BRM Images from the Oracle Software Delivery Website" in *BRM Cloud Native Deployment Guide*.

Table 1-3 lists the names of the packages and package files that you need to extract for using and configuring AI and ML services.

Table 1-3 Cloud Native Package and Package Files

Package Name	Package File Name
Oracle Communications Cloud Native ML Data Service Images 15.2.0	**ml-data-fetch.tgz** Includes: • **ml-data-fetch-orchestrator.tar** • **ml-data-fetch-processor.tar**
Oracle Communications Cloud Native ML Training Service Images 15.2.0	**ml-train-utility.tgz** Includes: • **ml-train-utility-orchestrator.tar** • **ml-train-utility-processor.tar**
Oracle Communications Cloud Native ML Recommendation Service Images 15.2.0	**ml-recommendation.tgz** Includes: • **ml-recommendation-orchestrator.tar** • **ml-recommendation-predictor.tar**

Monitoring AI Services in Cloud Native Environment

You can monitor all the AI services using various service metrics. These metrics allows you to track and monitor the total number of requests and the time taken by the services.

Table 1-4 lists the available metrics for the services along with their descriptions.

Table 1-4 Service Metrics

Metric Name	Description
aiml_data_service_request_seconds	Total time taken by the previous data service request.
aiml_data_service_request_seconds_count	Total number of data service requests received.
aiml_data_service_request_seconds_max	Maximum time taken by a data service request to process.
aiml_data_service_request_seconds_sum	Total time taken by all the data service requests to finish processing.
aiml_train_service_request_seconds	Total time taken by the previous training service request.

Table 1-4 (Cont.) Service Metrics

Metric Name	Description
aiml_train_service_request_seconds_count	Total number of training service requests received.
aiml_train_service_request_seconds_max	Maximum time taken by a training service request to process.
aiml_train_service_request_seconds_sum	Total time taken by all the training service requests to finish processing.
recommend_service_request_seconds	Total time taken by the previous prediction service request.
recommend_service_request_seconds_count	Total number of prediction service requests received.
recommend_service_request_seconds_max	Maximum time taken by a prediction service request to process.
recommend_service_request_seconds_sum	Total time taken by all the prediction service requests to finish processing.

In addition to these, all the metrics that are provided by Micrometer, Tempo, and Kubernetes are available.

Configuring Grafana for AI Services Metrics

You can create a dashboard in Grafana to display the metric data for your AI services.

Alternatively, you can use the sample dashboards included in the **ml-helmcharts.tgz** package.

Table 1-5 lists each sample dashboard.

Table 1-5 Sample Grafana Dashboards

File Name	Description
data-fetch-grafana-dashboard.json	Allows you to view metrics for the data service requests.
train-utility-grafana-dashboard.json	Allows you to view metrics for the training service requests.
recommendation-grafana-dashboard.json	Allows you to view metrics for the prediction or recommendation service requests.

2

About Prerequisite Software Requirements

Learn about the required operating system, software, tools, and files required to deploy artificial intelligence (AI) and machine learning (ML) services in Oracle Monetization Suite.

Topics in this document:

About Operating System Requirements

AI and ML services in Oracle Monetization Suite require a Linux operating system. For a list of supported Linux versions, see *BRM Compatibility Matrix*.

Setting Up Prerequisite Software and Tools

Before you deploy data, training, and prediction services, you must install and configure the following software and tools:

1. **Kubernetes Cluster**: Set up a Kubernetes cluster to orchestrate and manage your containerized service deployments.

   For more information about Kubernetes, see "Kubernetes Concepts" in the *Kubernetes documentation*.

2. **Container Runtime**: Install a Kubernetes-supported container runtime such as Podman on all cluster nodes to support containerized workloads.

3. **Helm 3**: Install Helm 3 to deploy and manage services using Helm charts.

   For installation instructions, see "Installing Helm" in the *Helm documentation*.

4. **Kubernetes Namespaces**: Create the following namespaces for separation of services:
   - **data-fetch-service** for the data service
   - **training-utility-service** for the training service
   - **recommend-service** for the prediction service

5. **Oracle Cloud Infrastructure (OCI) Policies**: If you plan to deploy services in OCI, create and assign necessary OCI policies to enable access between Object Storage, Logging, Data Flow, and Data Science.

   A policy defines who has access to specific OCI resources and what actions they can perform. For more information, see "Getting Started with Policies" in the *OCI documentation*.

> **Note**
>
> This step is required only for deployments in Oracle Cloud Infrastructure.

6. **Oracle Identity Cloud Service (IDCS)**: Obtain access to Oracle Identity Cloud Service (IDCS), which serves as an identity provider for user authentication and access management.

   For more information, see "Federating with Identity Providers" in the *OCI documentation*.
7. **Prometheus**: Install Prometheus to monitor service metrics and trigger alerts.
8. **Grafana**: Install Grafana for visualizing metrics, importing dashboards, and analyzing data.
9. **Tempo**: Install Tempo for distributed tracing to monitor and troubleshoot services.
10. **NGINX Ingress Controller**: Install the NGINX Ingress Controller.

    For installation instructions, see "Installation Guide" in the *Kubernetes documentation*.

About Software Compatibility

To ensure efficient deployment and operation, use the current, supported versions of all required software, tools, platforms, and reference files. For details and a list of supported versions, see *BRM Compatibility Matrix*.

3

Common Installation Tasks for Oracle Monetization Suite AI Services

Learn about the common tasks required to deploy data, training, and prediction services for artificial intelligence (AI) and machine learning (ML) in Oracle Cloud Infrastructure (OCI).

Topics in this document:

Overview of Installation Tasks

To install and deploy data service, training service, and prediction service, you need to perform some common installation tasks. These installation tasks are required only if you plan to deploy these services in the OCI environment. These tasks help you to set up the environment.

The following is the list of high-level installation tasks:

1. Create a bucket in OCI Object Storage to store configuration files and resources.

   See "Creating Object Storage Buckets in OCI".
2. Create a private endpoint in OCI Data Flow.

   See "Creating Data Flow Private Endpoints".
3. Generate and download public and private API keys.

   See "Generating API Keys".
4. Create and verify the **archive.zip** deployment file.

   See "Creating the Archive.zip File".
5. Create any required Python scripts to run in OCI Data Flow.

   > **Note**
   >
   > Complete this step only if you plan to use OCI Data Science for job creation.

6. Upload configuration and dependency files to OCI Object Storage.

   See "Uploading Files to Object Storage".

7. Create a Data Flow application in OCI.

   See "Creating Data Flow Application".

8. Create jobs in OCI Data Science as needed.

   See "Setting Up a Data Science Job".

> **Note**
>
> Complete this step only if you plan to use OCI Data Science for job creation.

9. Set up an Oracle Identity Cloud Service (IDCS) instance to manage authentication.

   See "Federating with Identity Providers" in the *OCI documentation*.

> **Note**
>
> Using IDCS is optional but highly recommended to handle authorization and authentication. It is configurable and all the provided services are certified with it.

Creating Object Storage Buckets in OCI

You need to create buckets in OCI Object Storage to store various configuration files, API keys, scripts, and other resources.

To create a bucket in OCI Object Storage:

1. From the Navigation menu, select **Storage**.

   The Storage window appears on the right pane.

2. Click **Buckets**.

   The Buckets page appears.

3. Click **Create Bucket**.

   The Create Bucket page appears.

4. Enter a name for your bucket and use the default values for other settings.

5. Click **Create Bucket**.

Creating Data Flow Private Endpoints

To create a private endpoint in OCI Data Flow:

1. From the Navigation menu, select **Analytics & AI**.

   The Analytics & AI window appears on the right pane.

2. Select **Data Flow**.

   The Data Flow page appears.

3. From the left navigation pane, select **Private Endpoints**.

   The Private Endpoints page appears.

4. Click **Create private endpoint**.

The Create private endpoint page appears.

5. Enter a name in the **Name** field.
6. From the list, select the VCN and Subnet if you plan to use a private network. For public network, you can leave this field empty.
7. In the **DNS zones to resolve** field, add the DNS whose servers need the access.

   Fill in additional fields as needed based on your requirements.
8. Click **Create**.

Generating API Keys

To generate and download API keys:

1. Click your profile in the OCI Console and select **User settings**.

   The User setting page appears.
2. In the left panel, select **API Keys**.

   The API Keys page appears.
3. Click **Add API Keys**.
4. Download both the **Private Key** and **Public Key** files.

> **Note**
>
> The **Private Key** file is used for all future authorization of OCI features.

Creating the Archive.zip File

To prepare the **archive.zip** deployment package:

1. Pull the dependency packer Docker image:

   ```
   docker pull phx.ocir.io/axmemlgtri2a/dataflow/dependency-packager-linux_arm64_v8:latest
   ```

> **Note**
>
> You can also use Podman, based on your preference.

2. Create a **requirements.txt** file listing required Python dependencies. For this use case, include **oci** and **pandas**.
3. Place required JAR files and configuration files in the same directory as **requirements.txt**.
4. Run the following command (using your Python version) to create **archive.zip**:

   ```
   docker run --platform linux/amd64 --rm -v $(pwd):/opt/dataflow --pull always -it phx.ocir.io/axmemlgtri2a/dataflow/dependency-packager-linux_x86_64:latest -p 3.11
   ```
5. Verify that **archive.zip** includes **ojdbc11.jar** and the configuration file.
6. Update the configuration file with your User OCID, Tenancy, Region, and other required values.

Uploading Files to Object Storage

To upload files to your Object Storage bucket:

1. In the OCI Console, search for and navigate to **Buckets** and select **Buckets** from the Services results.

   The Buckets page appears with a list of available buckets.
2. Select your bucket.
3. Click **Upload objects**.
4. In the upload wizard:
   a. In the **Select files** section:
      - (Optional) In **Object name prefix**, enter a prefix to prepend it to each uploaded file name.

        To organize files in folders or subfolders, start the prefix with a slash (“/”). For example, `/myfiles` stores files in a `myfiles` folder.
      - Select **Standard** for the storage tier field.
      - Drag your file to the upload area or click to select it.
      - Click **Next**.
   b. In the **Review files** section:
      - Confirm the details and files to upload.
      - Click **Next**.
   c. In the **Upload files** section:
      - Click **Upload objects**.
      - Wait for the upload to complete.
      - Click **Close** when the process finishes.
      - Verify that your file appears in the Objects list for the bucket.

Creating Data Flow Application

To create an application in OCI Data Flow, perform the following steps:

1. Go to the OCI Console.
2. From the Navigation menu, select **Analytics & AI**.

   The Analytics & AI window appears on the right pane.
3. In the Analytics & AI window, select **Data Flow**.

   The Data Flow page appears.
4. Click **Create Application**.

   The create application details page appears.
5. From the **Spark version** list, select the required version.

> **Note**
>
> Oracle recommends you to use the latest version of Spark for best results.

6. In the **Driver Shape** and **Executer Shape** fields, select the required shape or hardware template. It should have a minimum of 2 OCPU and 32 GB of memory.
7. In the **Number of Executors** field, select **1** or more.
8. In the **Language** field, select **Python**.
9. In **Select a file**, select your bucket and the **dataflow.py** file.

> **Note**
>
> You can locate the **dataflow.py** file in the artifacts provided with the package and you can configure this file as per your requirements.

10. In the **Archive URI** section, select your bucket and the **archive.zip** file.

    See "Creating the Archive.zip File" for more information.
11. In **Application log location**, select your bucket for log storage.
12. Click **Create**.

    The data flow application is created.

Setting Up a Data Science Job

> **Note**
>
> The steps for creating training jobs may vary. For details, see "Deploying the Training Service in OCI and Non-OCI Environments".

To create and set up a job in OCI Data Science, perform the following tasks:

1. Go to the OCI Console.
2. From the Navigation menu, select **Analytics & AI**.

   The Analytics & AI window appears on the right pane.
3. Select **Data Science**.

   The Data Science page appears.
4. Click **Create project**.

   The Create project page appears.
5. Enter the required **Name** and **Description** for your project, then click **Create**.
6. Upload the **oci_config** file, **private_key** file, and **data_flow_config** files to your object storage bucket.

> **Note**
>
> You can locate these files in the artifacts provided with the package and you can configure them as per your requirements.

For more information, see "Uploading Files to Object Storage".

7. Go to your project, select **Jobs**, and click **Create job**.
8. In the **Basic Information** section:
   a. Select **Single Node**.
   b. Choose the appropriate compartment.
   c. Enter the required **Name** and **Description** for your job.
9. In the **Configuration** section, set the following environment variables:
   a. **JOB_RUN_ENTRYPOINT**: **controller.py**
   b. **CONDA_ENV_SLUG**: **python_p312_any_x86_64_v1** (for the data service) or **tensorflow216_p310_gpu_v1** (for Training Service)
   c. **CONDA_ENV_TYPE**: **service**
10. In the **Compute Shape** section, select the required shape.

> **Note**
>
> You must choose a compute shape with at least 1 core CPU and 16 GB RAM.

11. In the **Storage** section, insert the required storage.

> **Note**
>
> You need to have at least 50 GB storage.

12. In the **Networking** section, select the appropriate VCN and subnet configurations if you plan to use a private network. For public network, you can leave this field empty.
13. In the **Upload job artifact** section, select the **collect_data_from_dataflow.zip** file.

> **Note**
>
> You can locate the **collect_data_from_dataflow.zip** file in the artifacts provided with the package.

14. In the **Additional Configuration** section, specify the log group and logs as required.
15. In the **Storage Mount** section, click **+ Add object storage mount**.
   a. Select the correct **Compartment** and **Bucket** from the list.
   b. Leave the **Object name prefix** blank.
   c. In the **Destination path and directory** field, enter **/home**.

16. Click **Create**.

    The data science job is created.

4

Deploying and Configuring the Data Service

Learn how to deploy and configure the data service for Oracle Monetization Suite in both Oracle Cloud Infrastructure (OCI) and non-OCI environments.

Topics in this document:

Deploying Data Service in OCI and Non-OCI Environments

You can deploy the data service in Oracle Cloud Infrastructure or in a non-OCI environment, depending on your requirements.

- **For deployments in Oracle Cloud Infrastructure:**

  Complete all installation prerequisites described in "Overview of Installation Tasks". Once the environment is prepared, install the data service Helm charts as described below.

- **For deployments in non-OCI environments:**

  Install the data service Helm charts as described in this section.

To deploy the data service using Helm charts:

1. Download the data service Helm charts from the provided deployment package.

   For more information, see "Downloading Packages for the Cloud Native Helm Charts and Docker Files" and "Setting Up Prerequisite Software and Tools".

2. Create a new key and generate an SSL certificate:

   ```
   openssl req -newkey rsa:2048 -nodes -keyout privateKeyName.key -x509 -days 365 -out certificateName.crt
   ```

3. Create a Kubernetes TLS secret using the generated certificate and private key:

   ```
   kubectl create secret tls secretName --cert=certificateName.crt --key=privateKeyName.key
   ```

4. Install the NGINX Ingress Controller:

   - Add the ingress-nginx Helm repository:

     ```
     helm repo add ingress-nginx
     ```

   - Update the Helm repository:

     ```
     helm repo update
     ```

   - Install the Ingress Controller:

     ```
     helm install ingress-nginx ingress-nginx/ingress-nginx --namespace nginxNamespace --create-namespace
     ```

5. Use the following command to attach your service to the NGINX controller (set appropriate values for parameters as required):

```
helm install ingress-nginx ingress-nginx/ingress-nginx --namespace
nginxNamespace \
   --set controller.service.enableHttp=false \
   --set controller.service.enableHttps=true \
   --set controller.service.ports.https=443 \
   --set controller.service.nodePorts.https=31231 nodePort \
   --set controller.config.ssl-redirect=true \
   --set controller.config.force-ssl-redirect=true \
   --set controller.ingressClassResource.name=ingressClassName \
   --set controller.ingressClass=ingressClassName \
   --create-namespace
```

6. Update the **values.yaml** file for your deployment:
   a. Set mount paths:
      - **dataFilesPath**: Mount path for data files if **tech_choice** is **spark** and **storage_type** is **pvc**.
      - **logFilesPath**: Mount path for log files. If not set, it defaults to **dataFilesPath**.

        > **Note**
        >
        > Ensure you create the necessary StorageClass (SC), PersistentVolume (PV), and PersistentVolumeClaim (PVC). Update the PVC name in the deployment configuration as required.

      - **artifactsPath**: Mount path for customizable Python scripts and configuration files. You may leave this blank if these files are in the data files directory.
      - Create separate PVs and PVCs for logs (**data-fetch-logs-pvc**) and, if needed, artifacts (**data-fetch-artifacts-pvc**).

        > **Note**
        >
        > These are container paths and do not need to match the host path.

      - Set folder permissions:

        ```
        groupadd -g 10001 oracle
        useradd -mr -u 10001 -g oracle oracle
        chown 10001:10001 -R mountPath
        ```

   b. Configure **imageRepository** and image tags for **dataFetchOrchestrator** and **dataFetchProcessor**.
   c. Specify the paths to place the following files in the mount path, under **dataFetchProcessor.configurableFiles**:
      - **filterFile**: Path to the file containing the Python script used for data filtering using **/fetch** request. Additional files can be imported as required.

- **dataSourceFile**: Path to the configuration file with information about data source and OCI connection.
- **ociConfigFile**: Path to the configuration file with OCI connection. Also, copy your **private.pem** file, downloaded while creating the API key, to this folder and mention its path.
- **logConfigFile**: Path to the configuration file used for any modifications required in the log level or format.

> **Note**
>
> A sample file is available for each of them in the artifacts provided with the package: **sample_filter_data.py** (for filterFile), **dnn_preprocess.py** (for additional files), **config.json** (for dataSourceFile), **oci_config** (for ociConfigFile), and **logging_config.py** (for logConfigFile). You can configure these files as per your requirements.

d. Set the host value to your deployment host's name.

e. Set **tlsSecretName** to the name creating a Kubernetes TLS secret.

f. Set **ingressClassName** to your ingress class name.

g. Set **identityURI**, **clientID**, and **clientSecret** to your IDCS configuration. To disable IDCS, set **dataFetchOrchestrator.idcs.enabled** to **false**.

h. Set **serviceMonitor.serviceNamespace** to the namespace where you have deployed the services.

7. Create SC, PV, and PVC resources in your namespace. In **pv-template.yaml**, update the **spec.hostPath.path** to the host system path that needs to be mounted:

```
kubectl apply -f helm/sc.yaml
kubectl apply -f helm/pv-template.yaml
kubectl apply -f helm/pvcTemplate.yaml
```

8. Update the **helm/cgiudatafetch/templates/deployment.yaml** file as required, especially the **spec.template.spec.volumes** section.

9. Install the Data Service Helm charts:

```
helm upgrade --install data-fetch-services helm/cgiudatafetch/ --
namespace=data-fetch-service
```

You can schedule the data fetching process at regular intervals. To do this, you need to configure a cron job to run the provided shell script as per your requirements.

> **Note**
>
> A sample script called **sample_bash.sh** is available for your reference in the artifacts provided with the package. This sample script sets the **delta_data** to **false** for the first time and sets it to **true** for all subsequent run.

For example, you can set up a cron job to run the script once a month. You can do this by performing the following steps:

1. Make the script executable:

   ```
   chmod +x /opt/scripts/sample_bash.sh
   ```

2. Edit the crontab:

   ```
   crontab -e
   ```

3. Run the following command to schedule the script to run at 2 AM on the first day of each month:

   ```
   0 2 1 * * /opt/scripts/sample_bash.sh >> /opt/scripts/sample_bash.log 2>&1
   ```

Configuring and Using the Data Service

You interact with the data service using REST APIs and can connect other systems through API calls.

For data service, there are three APIs that you need to use:

- **Collect API** (**/data/collect**)

  Collects data from specified sources, such as Billing and Revenue Management databases or your custom databases.

- **Fetch API** (**/data/fetch**)

  Retrieves collected data based on request parameters and delivers it to required clients.

- **Cache API** (**/data/cache**)

  Caches fetched data for fast, repeated access.

These APIs are designed to collect, cache, and make data available for downstream machine learning and analytical tasks.

For more information, see "About the REST APIs".

5

Deploying and Configuring the Training Service

Learn how to deploy and configure the training service for artificial intelligence (AI) and machine learning (ML) in Oracle Monetization Suite within Oracle Cloud Infrastructure (OCI) and non-OCI environments.

Topics in this document:

Deploying the Training Service in OCI and Non-OCI Environments

You can deploy the training service either in Oracle Cloud Infrastructure or in a non-OCI environment according to your requirements.

For OCI deployments:

- Complete all common installation tasks described in "Overview of Installation Tasks".
- When uploading files to Object Storage, also upload the following:
  - **OCI configuration file**
  - **Private key file**
  - **Preprocessing script** (Optional)

> **Note**
>
> A sample for each of these files are available as part of the artifacts provided with the deployment package.

- When setting up data science jobs, create the following five jobs:
  - **Data Fetch Job**

    Set JOB_RUN_ENTRYPOINT to **entry_file.py**

    Job artifact: **data_fetch_from_os.zip**
  - **Preprocess Job**

    Set JOB_RUN_ENTRYPOINT to **preprocess_entry.py**

    Job artifact: **preprocess.zip**
  - **Train Job**

    Set JOB_RUN_ENTRYPOINT to **train_entry.py**

Job artifact: **train.zip**

- **Artifact Job**

  Set JOB_RUN_ENTRYPOINT to **artifact_entry.py**

  Job artifact: **artifact.zip**

- **Deploy Job**

  Set JOB_RUN_ENTRYPOINT to **deployment_entry.py**

  Job artifact: **deploy.zip**

> **Note**
>
> Oracle provides all of these zip files as part of the artifacts given with the deployment package.

For non-OCI deployments: You only need to install the Helm charts for the training service as described below.

To install the training services using Helm charts:

1. Download the training utility service Helm charts from the deployment package.

   For more information, see "Downloading Packages for the Cloud Native Helm Charts and Docker Files" and "Setting Up Prerequisite Software and Tools".

2. Generate the SSL certificate and private key:

   ```
   openssl req -newkey rsa:2048 -nodes -keyout privateKeyName.key -x509 -days 365 -out certificateName.crt
   ```

3. Create a Kubernetes TLS Secret using your certificate and private key:

   ```
   kubectl create secret tls secretName --cert=certificateName.crt --key=privateKeyName.key
   ```

4. Install the Ingress NGINX controller:

   - Add the ingress-nginx Helm repository:

     ```
     helm repo add ingress-nginx
     ```

   - Update the Helm repository:

     ```
     helm repo update
     ```

   - Install the Ingress Controller:

     ```
     helm install ingress-nginx ingress-nginx/ingress-nginx --namespace nginxNamespace --create-namespace
     ```

5. Attach the service to the NGINX Controller:

   ```
   helm install ingress-nginx ingress-nginx/ingress-nginx --namespace nginxNamespace \
     --set controller.service.enableHttp=false \
     --set controller.service.enableHttps=true \
     --set controller.service.ports.https=443 \
     --set controller.service.nodePorts.https=31231 nodePort \
     --set controller.config.ssl-redirect=true \
     --set controller.config.force-ssl-redirect=true \
     --set controller.ingressClassResource.name=ingressClassName \
   ```

```
--set controller.ingressClass=ingressClassName \
--create-namespace
```

6. Update the **values.yaml** file for your deployment:
   a. Set the following mount paths:
      - **modelFilesPath**: Mount path for model files. This is required for both OCI and non-OCI deployments.
      - **dataFilesPath**: Mount path for data files if **storage_type** is **pvc**.
      - **logFilesPath**: Mount path for log files. If not set, it defaults to **modelFilesPath**.
      - **artifactsPath**: Mount path for Python scripts and configuration files, or leave blank if these files are in the model files directory.
      - Create separate PV and PVC as needed for each path (**train-artifacts-pvc**, **train-logs-pvc**, **data-storage-pvc**).

        > **Note**
        >
        > These paths exist inside the container and do not need to match the host path.

      - Set folder permissions:

        ```
        groupadd -g 10001 oracle
        useradd -mr -u 10001 -g oracle oracle
        chown 10001:10001 -R mountPath
        ```

   b. Set the following configuration parameters:
      - **imageRepository**
      - IMAGE_TAG for both **trainUtilityOrchestrator.image** and **trainUtilityPredictor.image**
   c. Specify the paths to place the following files in the mount path, under **trainUtilityPredictor.configurableFiles**:
      - **preprocessScript**: Path to the Python preprocessing script
      - **ociConfigFile**: Path to the configuration file with OCI connection details
      - **datascienceConfig**: Path to the configuration for data science jobs
      - **labelFile**: Path to the label file for DNN model training
      - **logConfigFile**: Path to the log configuration

        > **Note**
        >
        > A sample file is available for each of them in the artifacts provided with the package: **sample_preprocess.py** (for preprocessScript), **oci_config** (for ociConfigFile), **config.json** (for datascienceConfig), **sample_label.csv** (for labelFile), and **logging_config.py** (for logConfigFile). You can configure these files as per your requirements.

   d. Set the host value to your deployment host's name.

e. Set **tlsSecretName** to the value used when creating the Kubernetes TLS secret.

f. Set **ingressClassName** to your ingress class name.

g. Set **identityURI**, **clientID**, and **clientSecret** to your IDCS configuration. To disable IDCS, set **trainUtilityOrchestrator.idcs.enabled** to **false**.

h. Set **serviceMonitor.serviceNamespace** to the namespace where you have deployed the services.

7. Create StorageClass (SC), PersistentVolume (PV), and PersistentVolumeClaim (PVC) resources in your namespace. In **pv-template.yaml**, update **spec.hostPath.path** to the host system path that should be mounted:

```
kubectl apply -f helm/sc.yaml
kubectl apply -f helm/pv-template.yaml
kubectl apply -f helm/pvcTemplate.yaml
```

8. Update the **helm/cgiutrainingutility/templates/deployment.yaml** file as required, especially the **spec.template.spec.volumes** section.

9. Install the Helm charts:

```
helm upgrade --install training-utility-services helm/cgiutrainingutility/
--namespace=training-utility-service
```

Configuring and Using the Training Service

You interact with the training service using REST APIs, which enables integration with external systems.

For training service, you use the following REST service:

- **Train API (/utility/train**)

  Use this API to train a model based on the data acquired by the data service and the configured parameters.

For more information, see "About the REST APIs".

6

Deploying and Configuring the Prediction Service

Learn how to deploy and configure the prediction service for Oracle Monetization Suite in both Oracle Cloud Infrastructure (OCI) and non-OCI environments.

Topics in this document:

Deploying the Prediction Service in OCI and Non-OCI Environments

You can connect to a model on Oracle Cloud Infrastructure or on a non-OCI environment and deploy the prediction service independently, depending on your requirements. For this, you need to perform the installation steps described in "Overview of Installation Tasks", and then install the prediction service Helm charts by following the instructions below.

To install and deploy the prediction service using Helm charts:

1. Download the recommendation service Helm charts from the deployment package.

   For more information, see "Downloading Packages for the Cloud Native Helm Charts and Docker Files" and "Setting Up Prerequisite Software and Tools".

2. Generate the SSL certificate and private key:

   ```
   openssl req -newkey rsa:2048 -nodes -keyout privateKeyName.key -x509 -days 365 -out certificateName.crt
   ```

3. Create a Kubernetes TLS Secret using your certificate and private key:

   ```
   kubectl create secret tls secretName --cert=certificateName.crt --key=privateKeyName.key
   ```

4. Install the Ingress NGINX controller:

   - Add the ingress-nginx Helm repository:

     ```
     helm repo add ingress-nginx
     ```

   - Update the Helm repository:

     ```
     helm repo update
     ```

   - Install the Ingress Controller:

     ```
     helm install ingress-nginx ingress-nginx/ingress-nginx --namespace nginxNamespace --create-namespace
     ```

5. Attach the service to the NGINX Controller:

   ```
   helm install ingress-nginx ingress-nginx/ingress-nginx --namespace nginxNamespace \
   ```

```
--set controller.service.enableHttp=false \
--set controller.service.enableHttps=true \
--set controller.service.ports.https=443 \
--set controller.service.nodePorts.https=31231 nodePort \
--set controller.config.ssl-redirect=true \
--set controller.config.force-ssl-redirect=true \
--set controller.ingressClassResource.name=ingressClassName \
--set controller.ingressClass=ingressClassName \
--create-namespace
```

6. Update the **values.yaml** file for your deployment:
   a. Set the following mount paths:
      - **modelFilesPath**: Mount path for model files. This is required for both OCI and non-OCI deployments.
      - **logFilesPath**: Mount path for log files. If not set, it defaults to **modelFilesPath**.
      - **artifactsPath**: Mount path for Python scripts and configuration files (use a separate mount path if using PVC). Leave blank if not required.
      - Create distinct PV and PVC for artifacts (**predict-artifacts-pvc**, if **artifactsPath** is set) and logs (**predict-logs-pvc**).

        > **Note**
        >
        > These paths are container directories and need not match host system paths.

      - Set folder permissions:

        ```
        groupadd -g 10001 oracle
        useradd -mr -u 10001 -g oracle oracle
        chown 10001:10001 -R mountPath
        ```

   b. Set the following configuration parameters:
      - **imageRepository**
      - IMAGE_TAG for both **recommendationOrchestrator.image** and **recommendationPredictor.image**
   c. Specify file paths in **recommendationPredictor.configurableFiles**:
      - **modelPath**: Path for the base folder where models are stored. Supported: PVC, OCI bucket (must include oci://), or Data Science URI (must include https://).
      - **modelNames**: File path for the list of models to be loaded at startup. Update and restart the service if new models are added.
        - For the PVC path, follow the {Stage}@{Model_Name} format. The files with '@' are stored in MLFlow. The remaining files are required supporting files.
        - For Data Flow, follow the {Name}@{OCI_Path} format.
        - For Data Science, follow the {name}@{Predict_URI} format.

> **Note**
>
> You must include necessary mapper file paths in **modelNames** for k-Nearest Neighbors (KNN) or Cosine algorithms when using OCI Object Storage. Follow the {Name}@{OCI_Path} format.

- **ociConfigFile**: Path to configuration file with OCI connection details.
- **preprocessScript**: Path to data preprocessing script for prediction.
- **logConfigFile**: Path to logging configuration file.
- **datascienceConfig**: Path to configuration for data science job integration.

> **Note**
>
> A sample is available for these files in the artifacts provided with the package: **oci_config** (for ociConfigFile), **sample_predict_preprocess.py** (for preprocessScript), and **logging_config.py** (for logConfigFile). You can configure these files as per your requirements.

d. Set additional deployment parameters:
   - Hostname (**ingress.host**)
   - TLS secret (**ingress.tlsSecretName**)
   - Ingress class (**ingress.ingressClassName**)
   - IDCS configuration: Set **identityURI**, **clientID**, **clientSecret** as required. To disable IDCS, set **recommendationOrchestrator.idcs.enabled** to **false**.
   - Monitoring namespace (**serviceMonitor.serviceNamespace**)

7. Create the StorageClass, PersistentVolume, and PersistentVolumeClaim resources. Update the host system path in **pv-template.yaml** as needed:

```
kubectl apply -f helm/sc.yaml
kubectl apply -f helm/pv-template.yaml
kubectl apply -f helm/pvcTemplate.yaml
```

8. Update the deployment template as required. Edit **helm/cgiurecommendation/templates/deployment.yaml**, especially the **spec.template.spec.volumes** section.
9. Install the Helm charts:

```
helm upgrade --install recommend-services helm/cgiurecommendation/ --
namespace=recommend-service
```

Configuring and Using the Prediction Service

You can use and configure the prediction service using its REST API. The prediction service accepts API calls from other systems for inference and recommendation.

For the prediction service, you use the following REST service:

- **Prediction API (/recommend/predict**)

Use this API to generate predictions or recommendations for specific input data after models are trained and deployed.

> **Note**
>
> Interactive REST API documentation is available through the Swagger UI at the **/openapi/ui** endpoint.

7

REST API Reference for AI and ML

Learn about the REST APIs required for creating the Artificial Intelligence (AI) and Machine Learning (ML) workflow and how to use them for configuring different services.

Topics in this document:

About the REST APIs

The REST API for AI and ML services allows you to create data sources, train models on the configured data sources, and make predictions based on a customer's usage pattern. You use these APIs for applying and configuring different services.

All REST Endpoints

The following is a list of endpoints required for using AI and ML services.

- Create the Data Set

  Method: POST

  Path: `/data/collect`
- Send Specified Data to a Client

  Method: POST

  Path: `/data/fetch`
- Create a Cache Object

  Method: POST

  Path: `/data/cache`
- Train the Model

Method: POST

Path: `/utility/train`

- Create a Prediction

  Method: POST

  Path: `/recommend/predict`

Quick Start

Set up your environment and then use the REST API for AI and ML to make your first API call by performing these tasks.

Prerequisites

Table 7-1 lists the prerequisites for setting up your environment and making API calls.

Table 7-1 Prerequisites

Prerequisite	More Information
Install Oracle Monetization Suite applications	This refers to the application under Oracle Monetization Suite with which you want to integrate the AI services. For more information, you can see the installation guide for the respective product.
Install cURL	Use cURL

Send a Request

After you set up your REST API, you can send a request to ensure that your connection works.

Use cURL

The examples within this document use cURL to demonstrate how to access the REST API for AI and ML services in Oracle Monetization Suite.

Task 1: Install cURL

To connect securely to the server, you must install a version of cURL that supports SSL and provide an SSL certificate authority (CA) certificate file or bundle to authenticate against the CA certificate store, such as Verisign.

The following procedure demonstrates how to install cURL on a Windows 64-bit system:

1. In your browser, navigate to the cURL Releases and Downloads page at https://curl.se/download.html.
2. Locate the version of the cURL software that corresponds to your operating system, click the link to download the ZIP file, and then install the software.
3. Go to the cURL **CA Certs** page at https://curl.se/docs/caextract.html and then download the **ca-bundle.pem** SSL CA certificate bundle to the folder in which you installed cURL.

Task 2: Set Environment Variable for cURL

In a command window, set the cURL environment variable, CURL_CA_BUNDLE, to the location of your local CA certificate bundle. For example:

```
C:\curl> set CURL_CA_BUNDLE=ca-bundle.pem
```

For information about CA certificate verification using cURL, see https://curl.se/docs/sslcerts.html.

Task 3: Start cURL

Start cURL and specify one or more of the command-line options defined in the following table:

Table 7-2 cURL Options and Descriptions

cURL Option	Description
`-d @filename.json` `--data @filename.json`	Identifies the file containing the request body in JSON format on the local machine. Alternatively, you can pass the request body with `-d "{id=5,status='OK'}"`.
`-F @filename.json` `--form @filename.json`	Identifies form data, in JSON format, on the local machine.
-H --header	Defines one or more of the following: • Content type of the request document • Hostname and port number of your Oracle Services Communications Proxy (SCP) Authority Server
`-i` `--include`	Displays header information in the response.
`-X method` `--request method`	Indicates the type of request method (for example, GET or POST).

Authenticate

The API for AI and ML services uses OAuth 2.0 access tokens to authenticate requests from clients.

Before you can send requests to REST API services, you must acquire a valid OAuth access token. Then, your clients must pass the token in the header of every request sent to an API service (applicable only for Oracle Identity Cloud Service).

You can use either Oracle Identity Cloud Service or Oracle Access Management to set up authentication for your client requests. For more information, see "BRM REST Services Manager Security" in *BRM Security Guide*.

Send Requests

Use these guidelines when sending requests using the REST API for AI and ML services.

URL Structure

Here is the URL structure for the requests:

`apiRoot/resourcePath`

where:

- *apiRoot* is for accessing the HTTP Gateway server at either **http://***hostname***:***Port* or **https://***hostname***:***Port*.
- *resourcePath* is the path to the endpoint.

For example, the URL for fetching the data is:

http://*hostname***:***port***/data/fetch**

where:

- *hostname* is the URL for the application server.
- *port* is the port for the application server.

Supported Methods

You can perform various operations on a resource using standard HTTP/HTTPS method requests.

Table 7-3 shows the supported methods.

Table 7-3 Supported Methods

HTTP Method	Description
POST	Create the resource.

Media Types

The REST API for AI and ML services supports the following media type:

- `application/json`

Supported Headers

The REST API for AI and ML services supports the following headers that may be passed in the header section of the HTTP/HTTPS request or response.

Table 7-4 Supported Headers

Header	Description	Example
`Content-Type`	The media type of the body of the request. This is required for POST and PUT requests.	`Content-Type: application/json`

Table 7-4 (Cont.) Supported Headers

Header	Description	Example
`Authorization`	The type of authorization. The REST API for AI and ML servicdes uses the OAuth 2.0 bearer token framework.	`Authorization: Bearer YOUR_TOKEN` where *YOUR_TOKEN* is your client application's OAuth 2.0 access token.
`Accept`	The media types that the client wants to receive in the response.	`Accept: application/json`
`Host`	The domain name (and optionally the port number) of the server to which the request is being.	`Host: api.example.com`

Swagger URL

If you use Swagger, you can make calls to the API for AI and ML services at this URL:

apiRoot`/openapi/ui/index.html`

Reference

This is some additional reference information to help you work with the REST API for AI and ML services.

Status Codes

When you call any of the REST API operations for AI and ML services, the response returns one of the standard HTTP/HTTPS status codes defined in the following table.

Table 7-5 Status Codes

HTTP Status Code	Description
`200 OK`	The request was completed successfully.
`400 Bad Request`	The request could not be processed because it contains missing or invalid information (such as a validation odes on a parameter or a missing required value).
`401 Unauthorized`	The request is not authorized. The authentication credentials must be added or validated.
`403 Forbidden`	The user can't be authenticated. The user does not have authorization to perform this request.
`404 Not Found`	The request includes a resource URI that does not exist.
`405 Method Not Allowed`	The HTTP method specified in the request (`POST`) is not supported for this request URI.
`409 Conflict`	The request can't be completed due to a conflict with the current state of the target resource.

Table 7-5 (Cont.) Status Codes

HTTP Status Code	Description
500 Internal Server Error	The server encountered an unexpected condition that prevented it from fulfilling the request.
501 Not Implemented	The server either does not recognize the request method or is unable to process the request.
503 Service Unavailable	The server is not available at the moment because it is either overloaded or experiencing downtime.

Create the Data Set

Method: POST

Path: /data/collect

Description: Creates the data set. It collects data from your data source, such as the BRM database or your personal database, and stores it in an object or file.

This shows the cURL command for sending a Create the Data Set request:

```
curl -H “Content-Type: application/json” -H “Authorization: Bearer YOUR_TOKEN” -H
“Accept: application/json” -X POST -d @file.json 'http://hostname:port/data/collect'
```

where:

- *YOUR_TOKEN* is your client application's OAuth 2.0 access token.
- *file.json* is the JSON file that specifies the details of the data to collect.
- *hostname* is the URL for the application server.
- *port* is the port for the application server.

Request Body

The following shows the fields in the request body along with some sample data.

```
{
    "@type" : "CollectRequest",
    "data_source_type": [
        "oracledb"
    ],
    "tech_choice": ("spark" / "dataflow" / "datascience"),

      # Set this to true if you need to append the training data. Set to
false if you want to fetch complete data without appending
    "delta_data": ("true" / "false"),

    "storage_details": [
        {
            "@type": "StorageDetails",
            "path": "/mnt/data_services/",
            "output_file_type": ("json" / "csv" / "pickle"),
            "storage_type": ("pvc" / "object_storage")
```

```
        }
    ],
    "query_builder": [
        {
            "@type": "QueryBuilder",
            "queries": [
                {
                    "@type": "QueryList",
                    "name": "account_data",
                    "query": "select a.poid_id0 as acct_cd, a.currency as
crncy_cd, b.country as cntry_name, b.state as state_name, b.city as city_name
from account_t a left join account_nameinfo_t b on b.obj_id0 = a.poid_id0
where b.country <> 'null'"
                },
                {
                    "@type": "QueryList",
                    "name": "msisdn_data",
                    "query": "select s.account_obj_id0, s.poid_type, a.name
as msisdn from service_t s left join service_alias_list_t a on a.obj_id0 =
s.poid_id0 where a.rec_id = 0"
                },
                {
                    "@type": "QueryList",
                    "name": "balance_group_data",
                    "query": "select d.poid_id0 as bal_grp_cd,
d.ACCOUNT_OBJ_ID0 as acct_cd, b.rec_id2 as acct_bal_typ_cd, b.VALID_TO as
bal_expry_dt, e.VALID_FROM as bal_begin_dt, e.CURRENT_BAL as bal_amt,
e.GRANTED_BAL as orgnl_bkt_amt, e.GRANTOR_OBJ_ID0 as prod_sbrp_cd,
e.GRANTOR_OBJ_TYPE as prod_sbrp_typ_cd from bal_grp_t d left join
bal_grp_sub_bals_t e on e.obj_id0 = d.poid_id0 inner join (select rec_id2,
max(valid_to) as valid_to from bal_grp_sub_bals_t group by rec_id2) b on
e.rec_id2 = b.rec_id2 and e.valid_to = b.valid_to"
                },
                {
                    "@type": "QueryList",
                    "name": "service_data",
                    "query": "select ACCOUNT_OBJ_ID0 as acct_cd,
BAL_GRP_OBJ_ID0 as bal_grp_cd, poid_id0 as ser_srvc_cd, poid_type as
ser_srvc_typ_cd, name as  ser_srvc_name from service_t where poid_type not
like '%/pcm_client' and poid_type not like '%/admin_client'"
                },
                {
                    "@type": "QueryList",
                    "name": "purchased_product_data",
                    "query": "select ACCOUNT_OBJ_ID0 as acct_cd,
CYCLE_START_T as cycl_strt_dt, CYCLE_END_T as cycl_end_dt, PLAN_OBJ_ID0 as
prod_ofr_cd, PRODUCT_OBJ_ID0 as prod_spec_cd, POID_ID0 as prod_sbrp_cd,
PURCHASE_START_T as prod_sbrp_strt_dt, PURCHASE_END_T as prod_sbrp_end_dt,
SERVICE_OBJ_ID0 as ser_srvc_cd, DEAL_OBJ_ID0 as deal_cd from
purchased_product_t"
                },
                {
                    "@type": "QueryList",
                    "name": "product_data",
                    "query": "select poid_id0 as prod_spec_cd, name as
shrt_name, permitted from product_t"
```

```
                },
                {
                    "@type": "QueryList",
                    "name": "deal_data",
                    "query": "select d.poid_id0 as deal_cd, d.permitted,
d.name, p.PRODUCT_OBJ_ID0 as prod_spec_cd from deal_t d left join
deal_products_t p on p.obj_id0 = d.poid_id0"
                },
                {
                    "@type": "QueryList",
                    "name": "plan_data",
                    "query": "select distinct p.POID_ID0 as prod_ofr_cd,
p.NAME as prod_ofr_name, d.PERMITTED as srvc_typ_cd, pd.DEAL_OBJ_ID0 as
deal_cd from PLAN_T p left join PLAN_SERVICES_DEALS_T pd on pd.OBJ_ID0 =
p.POID_ID0 left join DEAL_T d on d.POID_ID0 = pd.DEAL_OBJ_ID0"
                },
                {
                    "@type": "QueryList",
                    "name": "rate_data",
                    "query": "select r.POID_ID0 as prod_ofr_price_cd,
r.RATE_PLAN_OBJ_ID0 as prod_ofr_price_typ_cd, rp.PRODUCT_OBJ_ID0 as
prod_spec_cd, rp.EVENT_TYPE as evt_typ_cd, rb.ELEMENT_ID as elmnt_cd,
rb.SCALED_AMOUNT as scaled_amt from rate_t r left join rate_plan_t rp on
rp.POID_ID0 = r.RATE_PLAN_OBJ_ID0 left join rate_bal_impacts_t rb on
rb.obj_id0 = r.POID_ID0"
                },
                {
                    "@type": "QueryList",
                    "name": "event_data",
                    "query": "select i.ACCOUNT_OBJ_ID0 as acct_cd,
i.OFFERING_OBJ_ID0 as prod_ofr_cd, i.PRODUCT_OBJ_ID0 as prod_spec_cd,
i.RESOURCE_ID as acct_bal_typ_cd, e.SERVICE_OBJ_TYPE as service_type,
sum(e.net_quantity) as usage from event_t e left join event_bal_impacts_t i
on i.OBJ_ID0 = e.POID_ID0 where i.OFFERING_OBJ_ID0 <> 0 group by
i.ACCOUNT_OBJ_ID0, i.OFFERING_OBJ_ID0, i.PRODUCT_OBJ_ID0, i.RESOURCE_ID,
e.SERVICE_OBJ_TYPE"
                },
                {
                    "@type": "QueryList",
                    "name": "deal_data_2",
                    "query": "select i.ACCOUNT_OBJ_ID0 as acct_cd,
i.OFFERING_OBJ_ID0 as prod_ofr_cd, p.DEAL_OBJ_ID0 as deal_cd from
event_bal_impacts_t i left join event_t e on i.OBJ_ID0 = e.POID_ID0 left join
purchased_product_t p on p.POID_ID0 = i.OFFERING_OBJ_ID0 where e.POID_TYPE =
'/event/billing/product/fee/cycle/cycle_forward_monthly' and
i.OFFERING_OBJ_ID0 <> 0"
                },
                {
                    "@type": "QueryList",
                    "name": "over_usage_data",
                    "query": "select i.ACCOUNT_OBJ_ID0 as acct_cd,
i.OFFERING_OBJ_ID0 as prod_ofr_cd, i.PRODUCT_OBJ_ID0 as prod_spec_cd,
EXTRACT(MONTH FROM (TO_DATE('01/01/1970', 'dd/mm/yyyy') + e.CREATED_T/86400))
as MONTH, EXTRACT(YEAR FROM (TO_DATE('01/01/1970', 'dd/mm/yyyy') +
e.CREATED_T/86400)) as YEAR, e.SERVICE_OBJ_ID0 as srvc_cd, sum(i.AMOUNT) as
amt, sum (e.NET_QUANTITY) / ABS(NULLIF(rb.SCALED_AMOUNT, 0)) as
```

```
USAGE_OVER_GRANT from event_bal_impacts_t i left join event_t e on i.OBJ_ID0
= e.POID_ID0 left join rate_plan_t rp on rp.PRODUCT_OBJ_ID0 =
i.PRODUCT_OBJ_ID0 left join rate_t r on rp.POID_ID0 = r.RATE_PLAN_OBJ_ID0
left join rate_bal_impacts_t rb on rb.obj_id0 = r.poid_id0 where
rp.event_type = '/event/billing/product/fee/cycle/cycle_forward_monthly' and
e.POID_TYPE like '/event/session%' and e.service_obj_type <> '/service/
pcm_client' and i.OFFERING_OBJ_ID0 <> 0 group by i.ACCOUNT_OBJ_ID0,
i.OFFERING_OBJ_ID0, e.SERVICE_OBJ_ID0, i.PRODUCT_OBJ_ID0, rb.SCALED_AMOUNT,
EXTRACT(MONTH FROM (TO_DATE('01/01/1970', 'dd/mm/yyyy') + e.CREATED_T/
86400)), EXTRACT(YEAR FROM (TO_DATE('01/01/1970', 'dd/mm/yyyy') + e.CREATED_T/
86400))"
                },
                {
                    "@type": "QueryList",
                    "name": "usage_details",
                    "query": "select i.ACCOUNT_OBJ_ID0 as acct_cd,
i.OFFERING_OBJ_ID0 as prod_ofr_cd, i.PRODUCT_OBJ_ID0 as prod_spec_cd,
i.RESOURCE_ID as acct_bal_typ_cd, EXTRACT(MONTH FROM (TO_DATE('01/01/1970',
'dd/mm/yyyy') + e.CREATED_T/86400)) as MONTH, EXTRACT(YEAR FROM
(TO_DATE('01/01/1970', 'dd/mm/yyyy') + e.CREATED_T/86400)) as YEAR,
e.SERVICE_OBJ_ID0 as srvc_cd, substr(e.SERVICE_OBJ_TYPE, instr
(e.SERVICE_OBJ_TYPE, '/', -1) + 1) as ser_srvc_name, sum (e.NET_QUANTITY) /
ABS(NULLIF(rb.SCALED_AMOUNT, 0)) as USAGE from event_bal_impacts_t i left
join event_t e on i.OBJ_ID0 = e.POID_ID0 left join rate_plan_t rp on
rp.PRODUCT_OBJ_ID0 = i.PRODUCT_OBJ_ID0 left join rate_t r on rp.POID_ID0 =
r.RATE_PLAN_OBJ_ID0 left join rate_bal_impacts_t rb on rb.obj_id0 =
r.poid_id0 where rp.event_type = '/event/billing/product/fee/cycle/
cycle_forward_monthly' and e.POID_TYPE like '/event/session%' and
i.RESOURCE_ID <> 840 and e.service_obj_type <> '/service/pcm_client' and
i.OFFERING_OBJ_ID0 <> 0 group by i.ACCOUNT_OBJ_ID0, i.OFFERING_OBJ_ID0,
e.SERVICE_OBJ_ID0, i.PRODUCT_OBJ_ID0, rb.SCALED_AMOUNT, e.SERVICE_OBJ_TYPE,
i.RESOURCE_ID, EXTRACT(MONTH FROM (TO_DATE('01/01/1970', 'dd/mm/yyyy') +
e.CREATED_T/86400)), EXTRACT(YEAR FROM (TO_DATE('01/01/1970', 'dd/mm/yyyy') +
e.CREATED_T/86400))"
                }
            ],
            # 'join_condition' is an optional input which can be used to join
the data from 2 queries
            "join_condition": [
                {
                    "@type": "JoinConditions",
                    "name": "joined_data",
                    "join": [
                        "service_data",
                        "purchased_product_data"
                    ]
                 },
                {
                       "@type": "JoinConditions",
                         "name": "left_join_1",
                    "join": [
                        "event_data",
                        "purchased_product_data"
                    ],
                          "join_type": "left",
                          "join_column": ["acct_cd_1", "acct_cd"]
```

```
                }
            ]
        }
    ]
}
```

Response Body

If successful, the response code 200 is returned with a response body. For example:

```
{
    "@type": "CollectResponse",
    "status": "Data Fetch Successful"
}
```

Send Specified Data to a Client

Method: POST

Path: /data/fetch

Description: Retrieves the specified data from your data set and sends it to the client.

This shows the cURL command for sending a Send Specified Data to Client request:

```
curl -H “Content-Type: application/json” -H “Authorization: Bearer YOUR_TOKEN” -H
“Accept: application/json” -X POST -d @file.json 'http://hostname:port/data/fetch'
```

where:

- *YOUR_TOKEN* is your client application's OAuth 2.0 access token.
- *file.json* is the JSON file that specifies the details of the data to collect.
- *hostname* is the URL for the application server.
- *port* is the port for the application server.

Request Body

The following shows the fields in the request body along with some sample data.

```
{
  "@type": "FetchRequest",
  "required_data": [
    "account_data",
    "over_usage_data",
    "deal_data",
    "purchased_product_data"
  ],

  # 'additional_data' is an optional field. This data can be used in custom
data filter script to filter out required information from cached data
  # If 'sample' is set to 'true', the sample_filter_data.py
  # script will be used. If it is set to 'false', then the sample script
configured in helm chart (helm_charts/values.yaml -
dataFetchProcessor.filterFile)
```

```
  # will be used to filter out data

  "additional_data": {
    "account_id": "2479678",
    "sample": ("true" / "false")
  }
}
```

Response Body

If successful, the response code 200 is returned with a response body. For example:

```
{
    "@type": "FetchResponse",
    "data": {
        "data": [(Data Requested)],
        "status": ("No data is stored currently" / "Successful")
    }
}
```

Create a Cache Object

Method: POST

Path: /data/cache

Description: Caches the specified data and uses it when required.

This shows the cURL command for sending a Create a Cache Object request:

```
curl -H “Content-Type: application/json” -H “Authorization: Bearer YOUR_TOKEN” -H
“Accept: application/json” -X POST -d @file.json 'http://hostname:port/data/cache'
```

where:

- *YOUR_TOKEN* is your client application's OAuth 2.0 access token.
- *file.json* is the JSON file that specifies the details of the data to collect.
- *hostname* is the URL for the application server.
- *port* is the port for the application server.

Request Body

The following shows the fields in the request body along with some sample data.

```
{
  "@type": "CacheRequest",
  "required_data": [
    "account_data",
    "msisdn_data",
    "balance_group_data",
    "service_data",
    "purchased_product_data",
    "product_data",
    "deal_data",
```

```
    "plan_data",
    "rate_data",
    "event_data",
    "deal_data_2",
    "over_usage_data",
    "usage_data"
  ],
  "storage_type": ("pvc" / "object_storage"),

  # In case the 'storage_type' is 'object_storage', then the
'additional_data' must have 'bucket' and 'namespace' while using Dataflow.
  # If the 'storage_type' is 'pvc', then the 'additional_data' must have
'path' which point to 'metadata.json' file

  "additional_data": {
    "bucket": "sample-for-dataflow",
    "namespace": "namespace"
  }
}
```

Response Body

If successful, the response code 200 is returned with a response body. For example:

```
{
    "@type": "CacheResponse"
    "status": "Data Caching Successful"
}
```

Train the Model

Method: POST

Path: /utility/train

Description: Trains the model based on the data acquired by the data service and the set configurations.

This shows the cURL command for sending a Train the Model request:

```
curl -H “Content-Type: application/json” -H “Authorization: Bearer YOUR_TOKEN” -H
“Accept: application/json” -X POST -d @file.json 'http://hostname:port/utility/train'
```

where:

- *YOUR_TOKEN* is your client application's OAuth 2.0 access token.
- *file.json* is the JSON file that specifies the details of the data to collect.
- *hostname* is the URL for the application server.
- *port* is the port for the application server.

Request Body

The following shows the fields in the request body along with some sample data.

```
{
    "@type": "TrainRequest",
    "algorithm": [
        {
            "@type": "Algorithms",
            "algo": "knn",
            "model_dir": "model/", # Accepts only alphanumeric and underscore
(_)
            "model_name": "recommend_knn",
            "hyper_parameters": {
                "k_value": 5
            }
        },
        {
            "@type": "Algorithms",
            "algo": "cosine",
            "model_dir": "model/", # Accepts only alphanumeric and underscore
(_)
            "model_name": "recommend_cosine",
            "hyper_parameters": {
                "k_value": 5
            }
        },
        {
            "@type": "Algorithms",
            "algo": "dnn",
            "model_dir": "model/", # Accepts only alphanumeric and underscore
(_)
            "model_name": "dnn",
            "hyper_parameters": {
                "test_size": 0.2, # Percentage of data that needs to be used
for validation. Optimal range is between 0.1 to 0.3
                "epochs": 150, # Number of epochs that the training should run
                "batch_size": 16,
                "loss": "categorical_crossentropy", # Loss function to be
used in training (mean_squared_error / categorical_crossentropy /
sparse_categorical_crossentropy / categorical_hinge / etc.). Refer keras loss
functions. https://www.tensorflow.org/api_docs/python/tf/keras/losses
            "optimizer": "rmsprop", # Optimizer used to reduce loss (adam /
rmsprop / adagrad / etc.). Refer keras optimizer functions https://
www.tensorflow.org/api_docs/python/tf/keras/optimizers.
                "learning_rate": 0.001 # Ideal range is between 0.0001 to 0.1
                "final_activation": "softmax" # The activation function that
must be used in the final layer ('softmax' / 'relu' / 'linear' / 'tanh' /
'sigmoid' etc.). https://www.tensorflow.org/api_docs/python/tf/keras/
activations
            }
        }
    ],
```

```
    # 'sample' must be 'true' when we need to use default preprocessing
script. If it is set to 'false',
    # then the sample script configured in helm chart (helm_charts/
values.yaml - trainUtilityProcessor.preprocessScript)

    "sample": true,

    "storage_details": [
        {
            "@type": "StorageDetails",
            "storage_type": "object_storage", # Possible values are
'object_storage' or 'pvc'

            # "required_data" values are the unique names given to queries
during data collect

            "required_data": [
                "account_data",
                "balance_group_data",
                "service_data",
                "purchased_product_data",
                "product_data",
                "deal_data",
                "plan_data",
                "rate_data",
                "event_data",
                "deal_data_2",
                "over_usage_data",
                "usage_details”
            ],

           # In case the 'storage_type' is 'object_storage', then the
'additional_details' must have 'bucket' and 'namespace' while using Dataflow.
For Datascience, 'additional_details' must have 'tech_choice' as
'datascience'
           # If the 'storage_type' is 'pvc', then the 'additional_details'
must have 'path', 'file_type' (json/csv/pickle), and 'model_stage' (Staging/
Production/Archived)

            "additional_details": {
                "bucket": "sample-for-dataflow",
                "namespace": "adcdefghijk"
            }
        }
    ]
}
```

Response Body

If successful, the response code 200 is returned with a response body. For example:

```
{
    "@type": "TrainResponse",
```

```
    "status": "Training Complete / Exception Stack trace"
}
```

Create a Prediction

Method: POST

Path: /recommend/predict

Description: Creates a prediction for an account using the specified algorithm.

This shows the cURL command for sending a Create a Prediction request:

```
curl -H “Content-Type: application/json” -H “Authorization: Bearer YOUR_TOKEN” -H
“Accept: application/json” -X POST -d @file.json 'http://hostname:port/recommend/predict'
```

curl -X POST 'http://*hostname***:***port***/recommend/predict' -d** *@file.json*

where:

- *YOUR_TOKEN* is your client application's OAuth 2.0 access token.
- *file.json* is the JSON file that specifies the details of the data to collect.
- *hostname* is the URL for the application server.
- *port* is the port for the application server.

Request Body

The following shows the fields in the request body along with some sample data.

```
{
  "@type": "RecommendRequest",

  # "msisdn": "8928292909" can also be used instead of 'account_id'
  "account_id": "2824533",

  # 'algorithms' can be 'dnn', 'knn', 'cosine' based on requirement
  "algorithms": ["knn", "dnn"]
}
```

Response Body

If successful, the response code 200 is returned with a response body. For example:

```
{
    "@type": "RecommendResponse",
    "@baseType": "string",
    "@schemaLocation": "string",
    "status": "Success",
    "recommendations": [
        {
            "@type": "RecommendationDetails",
            "algorithm": "knn",
            "account_id": "2824533",
            "present_deal": "Deal 10 - Data Storm",
```

```
                "top_results": [
                    {
                        "@type": "PredictDetails",
                        "prediction": "Deal 12 - Data Mini",
                        "probability_or_distance": 0.0
                    },
                    {
                        "@type": "PredictDetails",
                        "prediction": "Deal 4 - SMS Special",
                        "probability_or_distance": 0.0
                    },
                    {
                        "@type": "PredictDetails",
                        "prediction": "Deal 3 - CO for Voice 40 ",
                        "probability_or_distance": 0.0
                    }
                ]
            },
            {
                "@type": "RecommendationDetails",
                "algorithm": "dnn",
                "account_id": "2824533",
                "present_deal": "Deal 10 - Data Storm",
                "top_results": [
                    {
                        "@type": "PredictDetails",
                        "prediction": "Deal 1 - Data 30",
                        "probability_or_distance": 0.11470177
                    },
                    {
                        "@type": "PredictDetails",
                        "prediction": "Deal 11 - Data Blaze",
                        "probability_or_distance": 0.11470167
                    },
                    {
                        "@type": "PredictDetails",
                        "prediction": "Deal 10 - Data Storm",
                        "probability_or_distance": 0.11470162
                    }
                ]
            }
        ]
    }
```


> **Note**
>
> You can configure the response body based on your requirements.