Oracle® AI Database

Telemetry Streaming Developer's Guide

26ai
G36665-01
October 2025

Oracle AI Database Telemetry Streaming Developer's Guide, 26ai

G36665-01

Primary Authors: Jiji Thomas, (primary author)

Contributing Authors: Tulika Das, (contributing author)

Contributors: Parthasarathy Raghunathan, Ken Kunar, Shanu Gandhi, Kritika Singh, Shishir Mathur, , (contributor)

Contents

List of Tables

1
Introduction to Oracle AI Database Telemetry Streaming

Oracle AI Database Telemetry Streaming (hereinafter called Telemetry Streaming) is a time series database built on Oracle AI Database for metrics streaming. Telemetry Streaming is designed specifically to store, retrieve, and manage times series data. You can use Telemetry Streaming with Oracle AI Database to build your time series database applications.

These topics provide the conceptual understanding that is needed to get started with Telemetry Streaming.

1.1 Concepts

The following are some basic concepts used in Telemetry Streaming.

Metric

A metric is the name of a variable whose value you want to track over a specific period. Metric names are labels that help us uniquely identify the metric that we are tagging. CPU_USAGE is an example of a metric name. Other examples can be something like OFFICE_TEMP, STOCK_PRICE, or NETWORK_USAGE. A metric may also include additional metadata like tags. Each data point in a metric has a timestamp.

Metric Tags

Metric tags let you capture different instances of the same metric name. For example, CPU_USAGE{device_owner : "JACK"} is one time series and CPU_USAGE{device_owner : "BENJAMIN"} is another time series.

Metric Sample

A metric sample is the record of the value of a metric at a specific point in time. A metric sample tells you when the measurement was taken and what value was recorded. For example, at 2025-05-21 10:00:00, a metric sample might record that the value of office temperature is 21.5. A metric sample can be represented as a tuple of metric name and tags.

The following is an example in JSON format that shows a single measurement of CPU usage value at a specific time:

```
{
  "metric": "cpu_usage",
  "timestamp": "2025-05-21T10:00:00",
  "value": 45.3,
  "tags": {
    "host": "server-01",
    "device_owner: "Jack"
  }
}
```

Time Series

A time series is a collection of metric samples for a given metric, arranged in a chronological order. Each data entry in a time series is timestamped and represented in time-value pairs. Metrics derived from time series data help analyze how a variable value changes over time. These samples are collected over a period to observe trends, detect anomalies, and perform analysis. Time series data includes server metrics, application performance monitoring data, network data, sensor data, events, clicks, trades in a market, and many other types of analytical data. For example, your office room temperature metric could measure the room temperature once per minute to show the associated values in Celsius for every minute in a series, such as 25 at time T1, 26 at time T2, and 24 at time T3, and so on.

```
{
  "metric": "room_temperature",
  "tags": {
    "unit": "Celsius",
    "building": "XYZ",
    "room": "LivingRoom",
    "sensor_id": "sensor-001",
    "floor": "1"
  },
  "samples": [
    {
      "timestamp": "2025-07-01T10:00:00Z",
      "value": 25
    },
    {
      "timestamp": "2025-07-01T10:01:00Z",
      "value": 26
    },
    {
      "timestamp": "2025-07-01T10:02:00Z",
      "value": 26.5
    }
  ]
}
```

The following example shows an extract of a time series depicted in JSON format for CPU usage values recorded over an hour:

```
{
  "metric": "cpu_usage",
  "tags": {
   "host": "server-01",
   "device_owner: "Jack"
  },
  "samples": [
    {
    "timestamp": "2025-05-21T10:00:00Z",
    "value": 45.3
    },
    {
    "timestamp": "2025-05-21T10:01:00Z",
    "value": 46.1
    },
    {
    "timestamp": "2025-05-21T10:02:00Z",
    "value": 44.8
    }
  ]
}
```

Time Series Database

A Time Series Database (TSDB) is a database that is specifically designed for storing, retrieving, and managing time series data. A TSDB is therefore more efficient than a general-purpose database in handling time series data. TSDBs have different architectural design properties that make them very different from other databases. These include timestamp data storage and compression, data lifecycle management, data summarization, ability to handle large time series-dependent scans of many records, and time series-aware queries. A TSDB is optimized for large volumes of sequential writes, making it easy to ingest high-throughput streams without bottlenecks.

Every TSDB has these components:

- Ingest clients to stream time series data into TSDB
- Database engine for storage and query of time series data
- Query component used by visualization application that are external to TSDB for visualizing and analyzing time series data
- Data lifecycle management

Ingesting and Querying Metrics

Ingesting and querying are the two core operations for handling time series data. Ingesting means collecting and storing metric data into the time series database. This involves writing metric samples into the database. These data may be derived from sensors for IoT devices, application logs, or third-party agents.

Querying means retrieving and analyzing stored metric data from the time series database for the purposes of visualization, alerting, or analysis. You can use query languages such as SQL or PromQL (Prometheus' query language) for querying data.

Epoch Time

Epoch time is a way to represent time as a single number by calculating the number of seconds that have elapsed since January 1, 1970 at 00:00:00 UTC.

Workspace

Workspace is an Oracle-specific concept that is used in Telemetry Streaming.

In the context of Telemetry Streaming, workspace is a namespace for storing data. Each workspace enables you to separate out metrics data from a manageability point of view, enabling you to manage the users who ingest data and the users who query data for each workspace. For example, you may want to store the metrics received from data center 1 in one workspace, and store the metrics from data center 2 in another workspace. You can restrict data center 1 to only certain users, while data center 2 to other users. Hence, workspaces allow you to segregate the data along with the user privileges. You can create different workspaces and give privileges for each workspace to specific users.

User

A user in the context of a workspace can be one of the following:

- An administrator of the workspace
- An ingest user who can ingest metric data into a workspace
- A query user who can query metric data from a workspace

See Also

Managing a Workspace for more information about a workspace and workspace users.

1.2 Telemetry Streaming Overview

Telemetry Streaming is a Time Series Database (TSDB) built on Oracle AI Database to collect, store, and process time series data. Telemetry Streaming offers a comprehensive, turnkey solution for metrics streaming with Oracle AI Database.

Metrics Streaming is essential for modern enterprise use cases such as DevOps monitoring, asset tracking, and anomaly detection. While Oracle AI Database offers best-of-breed features for building a metrics streaming solution, assembling these components can be complex and time-consuming. Telemetry Streaming simplifies this process by packaging Oracle's powerful technologies into a streamlined, ready-to-use solution, enabling faster deployment, easier management, and reduced operational overhead for metrics streaming applications.

You can enable Oracle AI Database to run Telemetry Streaming on a basic setup with only the Telemetry Streaming PL/SQL packages installed. You also have the option of a full-fledged implementation of Telemetry Streaming with functionality that:

- Supports ultra-fast metrics ingestion using REST and PL/SQL
- Enables users to query metrics data for monitoring and alerting purposes through SQL, PL/SQL, or PromQL (Prometheus' Query Language)
- Automates data lifecycle management, including compression and downsampling for older metrics data, data retention, and optimizing storage and performance over time

1.3 Time Series in Telemetry Streaming

In Telemetry Streaming, time series is represented in the following manner.

- A typical time series is identified by a combination of a metric name, say CPU_SECONDS, and a set of tags. The tags are key-value pairs that are specified in the JSON format. For example: `{"server":"localhost", "ID":1, "make":"Intel"}`. This combination uniquely identifies a time series.
- Each time series is associated with an array of tuples <*value*, *time*> that signifies the value of the time series at the specified time.
- The value can be a float or an integer. The time is a float that specifies the time in seconds since epoch (1-Jan-1971 00:00 UTC). Fractional seconds can be used for sub-seconds.

- The value and time are immutable after they enter the system. They can only be deleted after the data retention time is over, but cannot be updated.

1.4 Data Format and Naming Conventions

Metric Data Format

In Telemetry Streaming, each metric data sample is canonicalized into the following set of information.

Table 1-1 Metric Data Format in Telemetry Streaming

Name	Data Type	Description
`METRIC_NAME`	`VARCHAR2(512)`	The metric name part of the time series.
`METRIC_TAGS`	`VARCHAR2(4000)`	The tags part of the time series in JSON format.
`METRIC_VALUE`	`NUMBER`	The point-in-time (PIT) value of the time series.
`METRIC_TIME_EPOCH`	`NUMBER`	The time in seconds since epoch.

Naming Conventions

PromQL queries should conform to the rules of the PromQL.

Table 1-2 Naming Conventions for PromQL Queries

Name	Format
Metric Names	Metric names can include ASCII letters, digits, and underscores. They must match the regular expression (regex): `[a-zA-Z_][a-zA-Z0-9_]*`.
Tag/Label Keys	Tag or label keys may consist of ASCII letters, digits, and underscores, and should match the following pattern: `[a-zA-Z_][a-zA-Z0-9_]*`.
Tag/Label Values	Tag or label values can be enclosed in either single or double quotes and may contain any character. The corresponding regex pattern to match a label value is: `'.*'`.

See Also

Querying Prometheus for more information about the basics of PromQL

2

Telemetry Streaming Architecture and Components

This chapter discusses the high-level architecture of Telemetry Streaming and its various components.

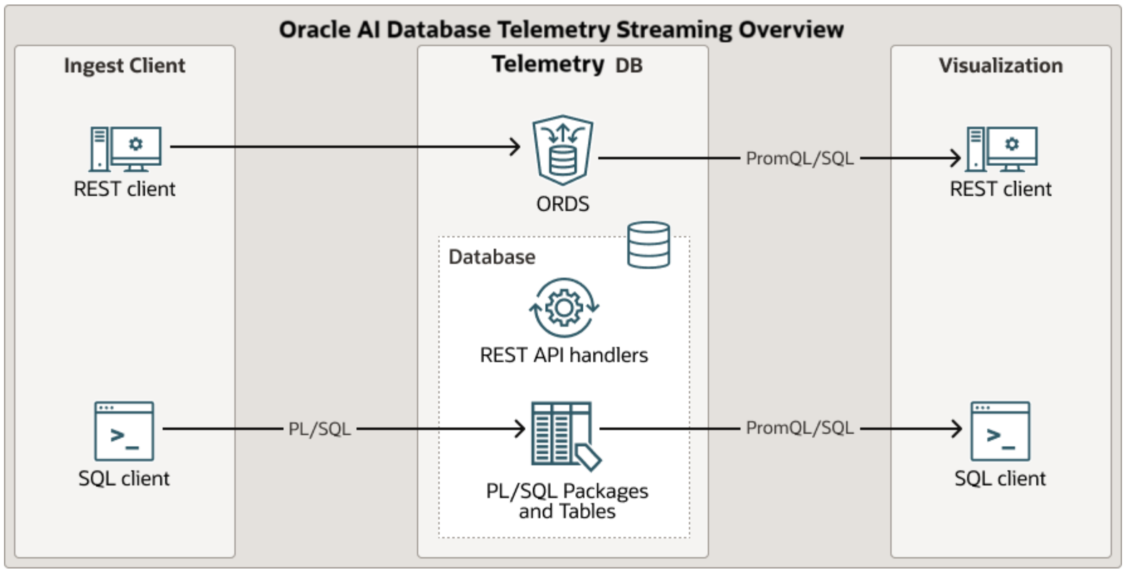
Topics:

- Telemetry Streaming Architecture
- Telemetry Streaming Components Overview

2.1 Telemetry Streaming Architecture

The following diagram shows a high-level architecture of Telemetry Streaming, providing an overview of the interconnect among its different components, with emphasis on the database components.

Figure 2-1 Telemetry Streaming Architecture



The PL/SQL packages and tables, and the REST API handlers are parts of the Telemetry Streaming implementation. Oracle REST Data Services (ORDS) is used to enable REST services for external clients.

The Telemetry Streaming components enable you to build robust metric streaming applications. Telemetry Streaming supports the push model of time series data ingestion, where clients ingest (push) their time series data into Telemetry Streaming. You can have SQL clients use the PL/SQL packages to ingest data into Telemetry Streaming. You can also have REST clients ingest metric data into Telemetry Streaming using REST APIs through ORDS.

You can have a REST client query Telemetry Streaming using PromQL or SQL. You can also use PromQL and SQL queries to query metrics on SQL clients.

You can find more details about these components in the following sections of this document.

2.2 Telemetry Streaming Components Overview

ORDS

Oracle REST Data Services (ORDS) needs to be installed with REST API handlers to enable external clients to use REST APIs and ingest data into or query data from Telemetry Streaming. The REST APIs are made available by initializing them using the `DBMS_TELEMETRY_ADMIN` PL/SQL package. All the REST APIs used through ORDS are authorized using the OAuth2 client credentials protocol.

Note

ORDS is not a part of Telemetry Streaming installation and needs to be installed separately.

Telemetry Streaming Database

Telemetry Streaming Database is a specialized database optimized for time series data. It is the central component that enables the storage, retrieval, and management of data. Ingest components, such as ORDS, push data into Telemetry Streaming for efficient storage and query.

Running the Telemetry Streaming SQL installation script enables an Oracle AI Database for the Telemetry Streaming service. Telemetry Streaming hosts the PL/SQL packages and tables that provide the Telemetry Streaming infrastructure.

3

Telemetry Streaming Administration

This chapter explains the administration and user management in Telemetry Streaming.

Topics:

- Administration Overview
  Learn in brief about the different types of users in Telemetry Streaming and their privileges.
- User Roles and Privileges
  This section explains the user roles in Telemetry Streaming and their associated privileges.
- Managing a Workspace
  This section explains how workspaces are managed in Telemetry Streaming.

3.1 Administration Overview

Learn in brief about the different types of users in Telemetry Streaming and their privileges.

Telemetry Streaming can have users that are existing database users or Oracle REST Data Services (ORDS) users.

Types of Database Users in Telemetry Streaming

Telemetry Streaming has 4 types of database users, based on their privileges and responsibilities:

1. TELEMETRY_DBA (also called Telemetry DBA), which is created automatically on Telemetry Streaming installation and owns the Telemetry Streaming infrastructure
2. Workspace, which is backed by an underlying schema and is used to logically separate Telemetry Streaming time series data for manageability
3. Workspace administrator, is a database user, who is designated by an Oracle DBA to one or more Telemetry Streaming workspaces for workspace administration
4. Telemetry Streaming users, are database users, who are assigned to workspaces by the workspace administrator for ingesting metric data, querying metric data, or both

See Also

Database User Types and Privileges for more information about the user roles and privileges of database users

Types of ORDS Users in Telemetry Streaming

Telemetry Streaming has 2 main types of ORDS users, based on their privileges and responsibilities:

1. ORDS Administrator user, who has administrative privileges for a workspace
2. ORDS Ingest or ORDS Query users, who have privileges to ingest data, query data, or both using ORDS endpoints

See Also

ORDS User Types and Privileges for more information about the user roles and privileges of ORDS users

3.2 User Roles and Privileges

This section explains the user roles in Telemetry Streaming and their associated privileges.

- Database User Types and Privileges
  The following table describes the different roles that each database user type plays in the Telemetry Streaming setup.
- ORDS User Types and Privileges
  The following table describes the different roles that each ORDS user type plays in the Telemetry Streaming setup.

3.2.1 Database User Types and Privileges

The following table describes the different roles that each database user type plays in the Telemetry Streaming setup.

Table 3-1 Database User Types and Responsibilities

User Type	Composition	Responsibility
TELEMETRY_DBA (also called Telemetry DBA)	One Per PDB	When Telemetry Streaming is installed, a Telemetry DBA user called TELEMETRY_DBA is also created. The Telemetry DBA owns all the PL/SQL packages that provide the Telemetry Streaming infrastructure. The Telemetry DBA is created with the Telemetry Streaming install script: `$ORACLE_HOME/admin/telemetry_install_plsql.sql`.
Workspace	Many per PDB	A workspace is used to logically separate time series data. Users can be assigned to each workspace for administration, ingestion, and querying. For each workspace that the customer creates, a passwordless database user (schema) is created that owns the data, metadata, and scheduler jobs for lifecycle management of data. The name of the passwordless schema is in the following format: `TM$<workspace name>`.

Table 3-1 (Cont.) Database User Types and Responsibilities

User Type	Composition	Responsibility
Telemetry Workspace Administrator (also called admin user)	Many per workspace	Any Oracle AI Database user with the DBA role can enable an existing PDB user to be the administrator for a particular workspace. Any Oracle DBA can assign appropriate privileges to the administrator by invoking the `dbms_telemetry_workspace.enable_workspace_admin` procedure. The workspace administrator can designate other existing PDB users to act as ingest or query users by using `dbms_telemetry_admin.enable_workspace_user` with appropriate arguments. The admin user has access to all the workspace administration PL/SQL packages and views.

See Also

- DBMS_TELEMETRY_ADMIN for more information about the workspace administration PL/SQL packages
- Tables and Views for more information about the workspace administration views

Table 3-1 (Cont.) Database User Types and Responsibilities

User Type	Composition	Responsibility
Telemetry Workspace Users	Many per workspace	An existing PDB user is enabled to be the user of a particular time series use case, which is represented as a workspace. A Telemetry workspace administrator can enable or disable existing database users using the PL/SQL APIs. A Telemetry workspace administrator for a particular workspace invokes the `dbms_telemetry_admin.enable_workspace_user` procedure to enable an existing PDB user for the workspace. These users can be assigned to ingest data, query data, or both. A PDB user can be assigned as an ingest user or query user to only one workspace at a time. The ingest user has access to the ingest package and the query user has access to the query package and the query user views. **See Also** • DBMS_TELEMETRY_INGEST for more information about the ingest user PL/SQL package • DBMS_TELEMETRY_QUERY and Tables and Views for more information about the query user PL/SQL package and views

3.2.2 ORDS User Types and Privileges

The following table describes the different roles that each ORDS user type plays in the Telemetry Streaming setup.

Table 3-2 ORDS User Types and Responsibilities

User Type	Composition	Responsibility
Telemetry ORDS Administrator (or ORDS Admin)	One per workspace	The Telemetry ORDS administrator is created for a workspace after the workspace is ORDS-enabled using `DBMS_TELEMETRY_ADMIN.ENABLE_WORKSPACE_ORDS`. The admin credentials (client ID and client secret) retrieved using `DBMS_TELEMETRY_ADMIN.GET_WORKSPACE_ORDS_ADMIN_AUTH` enables the ORDS admin user to carry out workspace administration tasks such as, adding ORDS ingest or query users, and viewing and manipulating parameters for their workspace.
Telemetry ORDS Users (ORDS Ingest Users and ORDS Query Users)	Many per workspace	The ORDS admin uses the relevant REST endpoints to add ORDS ingest or query users to a workspace. Based on their assigned roles, these ORDS users can then access the corresponding ingest or query REST endpoints using the OAuth credentials, which was generated at the time of creating these ORDS users to ingest, query, or both.

See Also

Managing ORDS Workspace Users for more information about how ORDS administrators and ORDS users are created

3.3 Managing a Workspace

This section explains how workspaces are managed in Telemetry Streaming.

Workspaces are managed differently for database users and ORDS users. The following sections explain the workspace management and administration procedures for database users and ORDS users.

Workspaces for Database Users Overview

The `DBMS_TELEMETRY_WORKSPACE` PL/SQL package deals with the workspace related operations. Each workpace is backed by an exclusive schema for storage of metric data. The name of the schema starts with TM$, such as `TM$<workspace_name>`. Any user with the Oracle DBA role can create or drop a workspace, and also enable or disable workspace administrators.

Within a workspace, there are 3 different roles that a user can play: a workspace administrator role, a query user role, or an ingest user role. The Oracle DBA can use the `DBMS_TELEMETRY_WORKSPACE` PL/SQL package to:

- Enable an existing database user as a workspace administrator
- Disable a user as a workspace administrator

The user with the workspace administrator role can then use the `DBMS_TELEMETRY_ADMIN` PL/SQL package or the workspace administration REST APIs to:

- Enable an existing database user as an ingest user, query user, or both for their workspace(s)

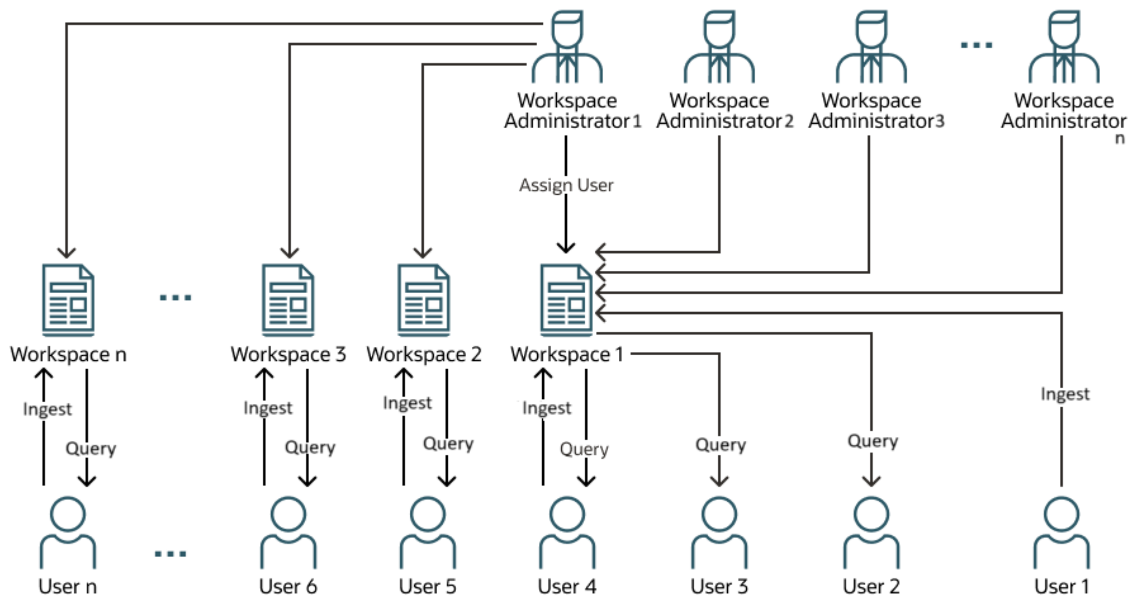
- Disable a user from their workspace(s)
- Set and get parameters for their workspace(s)

See Also

DBMS_TELEMETRY_WORKSPACE and DBMS_TELEMETRY_ADMIN in *Oracle AI Database PL/SQL Packages and Types Reference* for more information about the `DBMS_TELEMETRY_WORKSPACE` and `DBMS_TELEMETRY_ADMIN` PL/SQL packages

The Oracle DBA can assign multiple workspace administrators to manage one workspace or assign one workspace administrator to manage multiple workplaces. As for workspace users, a workspace administrator can assign multiple users to a workspace, but can assign one user to only one workspace, who can be an ingest user, a query user, or both for that workspace. The following diagram shows the relationship between workspaces and workspace administrators, and the relationship between workspaces and users.

Figure 3-1 Telemetry Streaming Workspace-User Relationships



Workspaces for ORDS Users Overview

You can find detailed information about managing ORDS workspace users and using REST API for workspace administration in Managing ORDS Workspace Users and Using REST API for Workspace Administration.

- Managing Database Workspace Users
  This section describes how different database user roles are managed in Telemetry Streaming.
- Managing ORDS Workspace Users
  This section describes how ORDS workspace users are managed in Telemetry Streaming.

3.3.1 Managing Database Workspace Users

This section describes how different database user roles are managed in Telemetry Streaming.

An Oracle DBA can create or drop workspaces. After a workspace is created, the Oracle DBA can assign the administrator role for a workspace to any existing database user, who is then called a workspace administrator.

The `DBMS_TELEMETRY_WORKSPACE` package allows the Oracle DBA to create or drop a workspace and enable or disable workspace administrators for the workspace. The Oracle DBA invokes the `DBMS_TELEMETRY_WORKSPACE.CREATE_WORKSPACE()` procedure to create a workspace and the `DBMS_TELEMETRY_WORKSPACE.ENABLE_WORKSPACE_ADMIN()` procedure to assign appropriate privileges to a workspace administrator.

As a part of the Telemetry Streaming installation and setup, the following management and administration procedures are used to manage database users in Telemetry Streaming:

Workspace Management

Any user with Oracle DBA privileges can manage a workspace using the following PL/SQL package.

Table 3-3 `DBMS_TELEMETRY_WORKSPACE`

Procedure/Function	Description
`CREATE_WORKSPACE`	Creates a new workspace
`DROP_WORKSPACE`	Drops a workspace
`ENABLE_WORKSPACE_ADMIN`	Adds an admin user to a workspace
`DISABLE_WORKSPACE_ADMIN`	Removes an admin user from a workspace

See Also

DBMS_TELEMETRY_WORKSPACE in *Oracle AI Database PL/SQL Packages and Types Reference* for more information about the `DBMS_TELEMETRY_WORKSPACE` PL/SQL package.

Additionally, the Oracle DBA has access to the following views.

Table 3-4 Workspace Management Views

Views	Description
`TELEMETRY_DBA.TELEMETRY_WORKSPACES`	Lists all the Telemetry Streaming workspaces created in the PDB
`TELEMETRY_DBA.TELEMETRY_ADMINS`	Lists all the admins associated with the Telemetry Streaming workspaces created in the PDB

See Also

Tables and Views for more information about the workspace management views

Workspace Administration

An existing Oracle AI Database user can be assigned as an administrator for a workspace. A workspace administrator can add users for ingestion or querying. The workspace administrator can also alter the workspace data life cycle management parameters.

A workspace administrator can use the following PL/SQL packages.

Table 3-5 `DBMS_TELEMETRY_ADMIN`

Procedure/Function	Description
`ENABLE_WORKSPACE_USER`	Adds an existing database user to a workspace
`DISABLE_WORKSPACE_USER`	Disables a user from the workspace
`SET_WORKSPACE_PARAMETER`	Sets a workspace parameter
`GET_WORKSPACE_PARAMETER`	Returns the value of a workspace parameter

See Also

DBMS_TELEMETRY_ADMIN in *Oracle AI Database PL/SQL Packages and Types Reference* for more information about the `DBMS_TELEMETRY_ADMIN` PL/SQL package.

Additionally, a workspace administrator has access to the following views.

Table 3-6 Workspace Administration Views

Views	Description
`TELEMETRY_DBA.TELEMETRY_WORKSPACE_USERS`	Lists all the users associated with all the workspaces administered by an Admin user
`TM$<workspace name>.TELEMETRY_INGEST_STATS`	Contains one row for every call made to ingest_metrics by the ingest users
`TM$<workspace name>.TELEMETRY_INGEST_DISCARDS`	Metrics whose time is older than 3600 seconds from the current time, are discarded and recorded in this view
`TM$<workspace name>.TELEMETRY_QUERY_STATS`	Contains one row for every call made to procedures of dbms_telemetry_query by the query users
`TM$<workspace name>.TELEMETRY_ADMIN_LOG`	Contains messages related to admin actions

See Also

Tables and Views for more information about the workspace management views

3.3.2 Managing ORDS Workspace Users

This section describes how ORDS workspace users are managed in Telemetry Streaming.

An ORDS workspace administrator can add ORDS users for ingestion or querying. The OAuth client ID:client secret pair of credentials, which is created by the ORDS admin user is required if the users want to push data into or query data from Telemetry Streaming through ORDS. The users then have to follow the OAuth 2-legged authorization mechanism to ingest or query data using ORDS endpoints.

To enable ORDS for a workspace and get the OAuth credentials of the ORDS workspace admin user, the following PL/SQL package is used.

Table 3-7 `DBMS_TELEMETRY_ADMIN`

Procedure/Function	Description
`ENABLE_WORKSPACE_ORDS`	This procedure enables ORDS for the given workspace and creates Telemetry Streaming REST endpoints
`GET_WORKSPACE_ORDS_ADMIN_AUTH`	This function returns the Telemetry Streaming ORDS admin user client-ID and cliend-secret OAuth credentials

The following steps are followed to set up a Telemetry Streaming workspace for ORDS users.

Before enabling ORDS for a workspace, the workspace schema (for example, `TM$WKSP1`) must be enabled from any `ORDS_DBA`.

```
exec ORDS.enable_schema(p_enabled => TRUE, p_schema => 'TM$WKSP1',
p_url_mapping_type => 'BASE_PATH',
 p_url_mapping_pattern => 'wksp1', p_auto_rest_auth => FALSE);
```

After the schema is ORDS-enabled, you must connect as the workspace admin and run the following statement:

```
exec dbms_telemetry_admin.enable_workspace_ords('wksp1');
```

Once the workspace is ORDS-enabled, an admin user is created. The admin credentials (client ID and client secret) can then be retrieved by running the following statement:

```
select dbms_telemetry_admin.get_workspace_ords_admin_auth('wksp1');
```

After this setup, the workspace is ORDS-enabled and ready for admin-authenticated interface usage.

Once an ORDS admin user is created for a workspace, ORDS ingest or query users can be added or removed using the `add_user` and `drop_user` REST endpoints (accessible by the admin user using the admin client ID-client secret).

A user can be added using the following endpoint:

```
/ords/<Workspace Name>/add_user
```

On successfully adding the user, the API returns the client ID and client secret of the new user.

These credentials allow the users to access relevant endpoints. For example, an ingest ORDS user `ing1` in workspace `wksp1` can access the following ingest endpoint:

```
/ords/wksp1/ing1/ingest
```

Users can be removed using the following endpoint:

```
/ords/<Workspace Name>/drop_user
```

Note

A single ORDS user can be granted both ingest and query privileges.

- Using REST API for Workspace Administration
  This section provides the documentation for REST APIs used to manage a workspace.

See Also

Using REST API for Workspace Administration for more information about the workspace administration APIs

3.3.2.1 Using REST API for Workspace Administration

This section provides the documentation for REST APIs used to manage a workspace.

Note

- Ensure that ORDS is installed and Telemetry Streaming is REST enabled before using REST APIs.
- To make REST API calls, you need an OAuth2 access token to use for authorization. The token can be derived after you provide the Client ID and Client Secret pair of credentials created by the ORDS workspace administrator using the PL/SQL interface.

See Also

Installing Oracle REST Data Services for more information about the ORDS installation.

ORDS Workspace Administration REST API

The ORDS workspace administration APIs enable the ORDS workspace administrator to perform a variety of administrative tasks, such as managing ORDS users, configuring and checking workspace parameters, and accessing workspace logs. There are several admin ORDS endpoints available for performing these administrative operations, such as adding new

ORDS users, removing existing users, and accessing or modifying configuration parameters for a workspace.

These admin endpoints are described in the following API documentation:

Table 3-8 REST API Summary

API Type	Summary
POST `/ords/<Workspace Name>/add_user`	Adds an ingest or query user to a workspace
POST `/ords/<Workspace Name>/drop_user`	Drops an ingest or query user from a workspace
GET `/ords/<Workspace Name>/list_users`	Gets the list of users: user names and roles in the current workspace
GET `/ords/<Workspace Name>/init_parameters`	Initializes the parameters to their default values
GET `/ords/<Workspace Name>/get_parameters`	Gets all the parameters with their values
POST `/ords/<Workspace Name>/set_parameter`	Sets a new value to a parameter
GET `/ords/<Workspace Name>/get_parameter/<Parameter Name>`	Gets the value of a given parameter
GET `/ords/<Workspace Name>/list_log/?from=<start time epoch>&to=<end time epoch>&limit=<number of entries>`	Scrapes all the top data (specified by `limit`) in `TELEMETRY_ADMIN_LOG` in the time duration range specified and shows as `collection_feed`
GET `/ords/<Workspace Name>/show_ingest_stats/?from=<start time epoch>&to=<end time epoch>&limit=<number of entries>`	Scrapes all the top data (specified by limit) in `telemetry_ingest_stats` in the time duration range specified and shows as `collection_feed`
GET `/ords/<Workspace Name>/show_query_stats/?from=<start time epoch>&to=<end time epoch>&limit=<number of entries>`	Scrapes all the top data (specified by limit) in `telemetry_query_stats` in the time duration range specified and shows as `collection_feed`

Add a User

METHOD:	**POST**
PATH:	`/ords/<Workspace Name>/add_user`
USER:	Workspace ORDS Admin User
SUMMARY:	Adds an ingest or a query user to a workspace
DATA:	`{"user_name":<User Name> , "role": <INGEST|QUERY>}`
HTTP CODE:	201 - Success with clientId and secret 400 - Failed with error message `{"error_code": <code>, "error_message":<message>}` 401 - Unauthorized

Add a User	
EXAMPLE:	PATH: `/ords/workspace1/add_user` DATA: `{"user_name":"INGEST1" , "role": "INGEST"}` CODE: 201 - Successfully inserted `{"client_id":"RWRvoD2byhqNdWImmWbelA..","client_secret":"jjkvQPAZbbVTg6iozelk0Q.."}` **Note** You can add a user with both ingest and query role in ORDS by specifying the "role" as "`INGEST|QUERY`" or "`QUERY|INGEST`".

Drop a User	
METHOD:	**POST**
PATH:	`/ords/<Workspace Name>/drop_user`
USER:	Workspace ORDS Admin User
SUMMARY:	Drops an ORDS ingest or an ORDS query user from a workspace
DATA:	`{"user_name":<User Name>}`
HTTP CODE:	201 - User dropped successfully: <User Name> 400 - Failed with error message `{"error_code": <code>, "error_message":<message>}` 401 - Unauthorized
EXAMPLE:	PATH: `/ords/workspace1/drop_user` DATA: `{"user_name":"INGEST1"}` CODE: 201 - User dropped successfully: INGEST1

LIST USERS	
METHOD:	**GET**
PATH:	`/ords/<Workspace Name>/list_users`
USER:	Workspace ORDS Admin User
SUMMARY:	Gets the list of user names, their respective roles, and the client ID and client secret of the users in the current workspace
HTTP CODE:	200 - Success with data `{[{"user_name": <user>,"role": <role>},...]}` 400 - Failed with error message `{"error_code": <code>, "error_message":<message>}` 401 - Unauthorized

LIST USERS	
EXAMPLE:	PATH: `/ords/workspace1/list_users` CODE: 200 - Success

```
{
    "items":[
        {
            "name":"INGEST1",
            "description":"A client for ingest management",
            "client_id":"_sPBlyIrAXpRLn6rRX4-YQ..",
            "client_secret":"C2e33aDKGM0-AowcgPjmmg.."
        },
        {
            "name":"admin_client",
            "description":"default admin client",
            "client_id":"CCpAyUsiCggqZNMrm4l3ZA..",
            "client_secret":"KPQF3kDOCClFanMDCS1Ugg.."
        }
    ],
    "hasMore":false,
    "limit":20,"offset":0,
    "count":2
}
```

Initialize Parameters	
METHOD:	**GET**
PATH:	`/ords/<Workspace Name>/init_parameters`
USER:	Workspace ORDS Admin User
SUMMARY:	Initializes the parameters to default values
HTTP CODE:	200 - Success, params initialized to default values 400 - Failed with error message {`"error_code": <code>, "error_message":<message>`} 401 - Unauthorized
EXAMPLE:	PATH: `/ords/workspace1/init_parameters` CODE: 200 - Success

Get Parameters	
METHOD:	**GET**
PATH:	`/ords/<Workspace Name>/get_parameters`
USER:	Workspace ORDS Admin User
SUMMARY:	Gets all the parameters with their values

Get Parameters	
HTTP CODE:	200 - Success `{[{"parameter_name": <name>,"parameter_value": <value>, "parameter_modified_timestamp":<modified date>},...]}` 400 - Failed with error message `{"error_code": <code>, "error_message":<message>}` 401 - Unauthorized
EXAMPLE:	PATH: `/ords/workspace1/get_parameters` CODE: 200 - Success

```
{
    "items":[

{"parameter_name":"delete_after_duration_hours","parameter_value":"2
40","parameter_modified_timestamp":"2025-07-04T11:17:38.381Z"},

{"parameter_name":"compress_after_duration_hours","parameter_value":
"24","parameter_modified_timestamp":"2025-07-04T11:17:38.392Z"},

{"parameter_name":"downsample_after_duration_hours","parameter_value
":"24","parameter_modified_timestamp":"2025-07-04T11:17:38.398Z"},

{"parameter_name":"downsample_interval_seconds","parameter_value":"6
0","parameter_modified_timestamp":"2025-07-04T11:17:38.400Z"},

{"parameter_name":"downsample_method","parameter_value":"avg","param
eter_modified_timestamp":"2025-07-04T11:17:38.404Z"}
    ],
    "hasMore":false,"limit":20,"offset":0,"count":5
}
```

SET Parameter Value	
METHOD:	**POST**
PATH:	`/ords/<Workspace Name>/set_parameter`
USER:	Workspace ORDS Admin User
SUMMARY:	Sets a new value to a parameter
DATA:	`{"parameter_name":<param_name>, "value":<new_value>}`
HTTP CODE:	201 - Success, parameter:param_name value set to:new_value 400 - Failed with error message `{"error_code": <code>, "error_message":<message>}` 401 - Unauthorized
EXAMPLE:	PATH: `/ords/workspace1/set_parameter` DATA: `{"parameter_name":"downsample_interval_seconds", "value":70}` CODE: 201 - Success `{"parameter":"downsample_interval_seconds","value":"70"}`

Get Parameter Value

METHOD:	**GET**
PATH:	`/ords/<Workspace Name>/get_parameter/<Parameter Name>`
USER:	Workspace ORDS Admin User
SUMMARY:	Gets the value for a given parameter
HTTP CODE:	200 - Success with data `{"parameter_name": <name>,"parameter_value": <value>, "default":<value>, last_modified: "timestamp"}` 400 - Failed with error message `{"error_code": <code>, "error_message":<message>}` 401 - Unauthorized
EXAMPLE:	PATH: `/ords/workspace1/get_parameter/downsample_interval_seconds` CODE: 200 - Success

```
{"parameter_name":"downsample_interval_seconds",
"parameter_value":70,
 "default":60, "last_modified":"04-JUL-25 12.00.06.542000 PM"}
```

Get Admin Logs

METHOD:	**GET**
PATH:	`/ords/<Workspace Name>/list_log/?from=<start time epoch>&to=<end time epoch>&limit=<number of entries>`
USER:	Workspace ORDS Admin User
SUMMARY:	This API scrapes all the top data (specified by limit) in `TELEMETRY_ADMIN_LOG` in the time duration range specified and shows as `collection_feed`.
HTTP CODE:	200 - Success with data 400 - Failed with error message `{"error_code": <code>, "error_message":<message>}` 401 - Unauthorized
EXAMPLE:	PATH: `/ords/workspace1/list_log/?from=1704700084&to=1704710884&limit=1` CODE: 200 - Success

```
{
   "items":[
       {
           "log_message":"DOWNSAMPLE:Checking Table_flat
Done: .040007",
           "log_timestamp":"2025-07-04T12:04:30.173Z"
       }
   ],
   "hasMore":false,"limit":0,"offset":0,"count":1
}
```

Get Ingest Data	
METHOD:	**GET**
PATH:	`/ords/<Workspace Name>/show_ingest_stats/?from=<start time epoch>&to=<end time epoch>&limit=<number of entries>`
USER:	ORDS Ingest User
SUMMARY:	This API scrapes all the top data (specified by limit) in `TELEMETRY_INGEST_STATS` in the time duration range specified and shows as `collection_feed`
HTTP CODE:	200 - Success with data 400 - Failed with error message `{"error_code": <code>, "error_message":<message>}` 401 - Unauthorized
EXAMPLE:	PATH: `/ords/workspace1/show_ingest_stats/?from=1704700084&to=1704710884&limit=100` OUTPUT:

```
{
    "items":[
        {
            "payload_size":559,
            "metrics_ingested":6,
            "metrics_given":6,
            "ingest_user_name":"INGEST1",
            "ingest_duration_milliseconds":1100,
            "ingest_method":"ords",
            "ingest_timestamp":"2025-07-03T09:21:05.236Z"
        }
    ],
    "hasMore":false,
    "limit":0,
    "offset":0,
    "count":2
}
```

Get Query Data	
METHOD:	**GET**
PATH:	`/ords/<Workspace Name>/show_query_stats/?from=<start time epoch>&to=<end time epoch>&limit=<number of entries>`
USER:	ORDS Query User
SUMMARY:	This API scrapes all the top data (specified by limit) in `TELEMETRY_QUERY_STATS` in the time duration range specified and shows as `collection_feed`
HTTP CODE:	200 - Success with data 400 - Failed with error message `{"error_code": <code>, "error_message":<message>}` 401 - Unauthorized

Get Query Data	
EXAMPLE:	PATH: `/ords/workspace1/show_query_stats/?from=1751636250&to=1751636258&limit=1` OUTPUT: `{ "items":[{ "query_type":"promql_instant", "start_time_epoch":1751636257, "end_time_epoch":null, "step_size_seconds":null, "query_user_name":"QUERYUSER1", "query_duration_milliseconds":380, "query_method":"ords", "query_timestamp":"2025-07-04T13:43:12.590Z" }], "hasMore":false,"limit":0,"offset":0,"count":1 }`

```
{
    "items":[
        {
            "query_type":"promql_instant",
            "start_time_epoch":1751636257,
            "end_time_epoch":null,
            "step_size_seconds":null,
            "query_user_name":"QUERYUSER1",
            "query_duration_milliseconds":380,
            "query_method":"ords",
            "query_timestamp":"2025-07-04T13:43:12.590Z"
        }
    ],
    "hasMore":false,"limit":0,"offset":0,"count":1
}
```

4

Installing and Enabling Telemetry Streaming in Oracle AI Database

This chapter describes the installation process and shows you how to enable Telemetry Streaming in Oracle AI Database.

Topics:

- Installation Overview
  Learn about the types of Telemetry Streaming installation and the installation roadmap.
- System Requirements
  This section lists the software and database features required for installing Telemetry Streaming.
- Enabling Telemetry Streaming in Oracle AI Database
  Learn how you can enable Telemetry Streaming in Oracle AI Database.
- Installing Oracle REST Data Services
  Learn about how to install Oracle REST Data Services.

4.1 Installation Overview

Learn about the types of Telemetry Streaming installation and the installation roadmap.

Note

Telemetry Streaming is available starting Oracle AI Database 26ai, Version 23.26.0, but it does not come bundled with Oracle AI Database. You must install Telemetry Streaming separately.

Basic Telemetry Streaming

As a minimal setup, if you want to run Telemetry Streaming using a SQL Plus client or any Oracle Call Interface (OCI) client, you can enable Oracle AI Database with Telemetry Streaming by running the SQL installation script from the admin directory, which installs the necessary PL/SQL packages for basic Telemetry Streaming setup. Once the script has run successfully, your database is Telemetry Streaming-enabled. You can then use the Telemetry Streaming PL/SQL packages for user administration, and ingest and query time series data.

See Also

Enabling Telemetry Streaming in Oracle AI Database for more information about the SQL installation script.

End-to-End Telemetry Streaming

An end-to-end Telemetry Streaming installation enables you to:

- Ingest metrics using REST or PL/SQL.
- Query metrics using SQL and PromQL through PL/SQL packages.

Table 4-1 Installation Roadmap

Installation Step	Required/ Optional	Purpose
Step 1: Run the SQL Script for installing the Telemetry Streaming packages.	Required	For enabling Telemetry Streaming on Oracle AI Database for basic use on SQL or OCI clients
Step 2: Install ORDS	Optional	For ingesting or querying metrics from external REST clients using REST APIs

4.2 System Requirements

This section lists the software and database features required for installing Telemetry Streaming.

The following table lists the software and their versions required for an end-to-end implementation of Telemetry Streaming.

Table 4-2 Software Requirements for Telemetry Streaming

Software	Purpose	Version
Oracle AI Database	For storing and retrieving metrics	Oracle AI Database 26ai, 23.26.0, or greater
ORDS	For REST API	ORDS version 23, or greater

Database Features

Telemetry Streaming is built on existing Oracle features and hence depends on these features in Oracle AI Database.

Table 4-3 Dependencies and Prerequisites

Feature	Purpose
Interval Partition	For data organization that helps in data management and query performance
Compression	For compression data beyond a threshold to reduce storage footprint
Scheduler Jobs	For data lifecycle management

4.3 Enabling Telemetry Streaming in Oracle AI Database

Learn how you can enable Telemetry Streaming in Oracle AI Database.

To enable Telemetry Streaming in Oracle AI Database, you must run the script from the following files, which are located in the `$ORACLE_HOME/rdbms/admin` directory.

> **Note**
>
> If you intend on using ORDS for REST API, ensure that ORDS is installed before running the ORDS-related script.

Table 4-4 Telemetry Streaming Installation Script Files

File	Functionality
`telemetry_install_plsql.sql`	This script creates the TELEMETRY_DBA user and loads all Telemetry Streaming PL/SQL packages under it. It also creates the required synonyms and metadata tables in both SYS and TELEMETRY_DBA schemas.
`telemetry_install_ords.sql`	This script loads all ORDS-related Telemetry Streaming PL/SQL packages (`prvthtelemetry_ords.plb` and `prvtbtelemetry_ords.plb`) into TELEMETRY_DBA.

Installation for Using Telemetry Streaming with PL/SQL

Complete the following steps to install Telemetry Streaming with a simple workspace setup having ingest, query and workspace admin users.

1. Run as SYS `@$ORACLE_HOME/rdbms/admin/telemetry_install_plsql.sql` (This creates TELEMETRY_DBA and installs Telemetry Streaming).
2. Create a tablespace that you want to use for Telemetry Streaming (say, `TMTBS`).
3. Create ingest, query, and admin users that you want to use for the workspace (say, `wrkspace_ingest_user`, `wrkspace_query_user`, and `wrkspace_admin_user`) and make the tablespace (`TMTBS`) you created in the previous step as their default tablespace for the workspace.
4. Create a workspace `WKSP1` on the `TMTBS` tablespace connecting as any database user with DBA privileges. Run the following statement.

   ```
   exec
   TELEMETRY_DBA.dbms_telemetry_workspace.create_workspace('WKSP1','TMTBS');
   ```

5. Connect as any database user with DBA privileges and make `wrkspace_admin_user` the admin of `WKSP1`. Run the following statement.

   ```
   exec
   TELEMETRY_DBA.dbms_telemetry_workspace.enable_workspace_admin('WKSP1','wrks
   pace_admin_user');
   ```

6. Connect as the Admin user `wrkspace_admin_user` and enable ingest and query users on workspace. Run the following statements.

```
exec
dbms_telemetry_admin.enable_workspace_user('WKSP1','wrkspace_ingest_user','
ingest')


exec
dbms_telemetry_admin.enable_workspace_user('WKSP1','wrkspace_query_user','q
uery')
```

The preceding steps (Installation for Using Telemetry Streaming with PL/SQL) conclude the Telemetry Streaming installation without ORDS. For installing Telemetry Streaming with ORDS, see Installing Oracle REST Data Services.

- Telemetry Streaming PDB Parameters
  Configure Telemetry Streaming PDB parameters to manage performance and efficiency of Telemetry Streaming.

See Also

DBMS_TELEMETRY_WORKSPACE and DBMS_TELEMETRY_ADMIN in *Oracle AI Database PL/SQL Packages and Types Reference* for more information about the workspace administration PL/SQL packages

4.3.1 Telemetry Streaming PDB Parameters

Configure Telemetry Streaming PDB parameters to manage performance and efficiency of Telemetry Streaming.

During the time of installation or after the installation, you can configure a few PDB parameters that best fit your use case. All parameters have their respective default values.

Table 4-5 Telemetry Streaming PDB Parameters

Parameter	Purpose	Default	Valid Values
`delete_after_duration_hours`	Delete data older than these many hours	240	1 to 87600
`compress_after_duration_hours`	Compress data older than these many hours	24	1 to 87600
`downsample_after_duration_hours`	Down sample data older than these many hours	168	1 to 87600
`downsample_interval_seconds`	Down sample the data in chunks of this interval in seconds	60	1 to 2600000
`downsample_method`	Aggregate method to be used to down sample the data in the interval	avg	avg, min, max, sum

4.4 Installing Oracle REST Data Services

Learn about how to install Oracle REST Data Services.

> **Note**
>
> Oracle REST Data Services (ORDS) must be installed separately. It is not included with Telemetry Streaming installation.

ORDS enables external clients to use REST APIs to ingest data into and query data from Telemetry Streaming.

To install ORDS, follow the installation instructions from the following link:

Installing and Configuring Oracle REST Data Services

Enabling Telemetry Streaming with ORDS

After the ORDS installation, run the SQL script to install the REST API handlers and enable the PL/SQL packages to be used with ORDS. The REST API infrastructure gets installed in Telemetry Streaming and you can start using the Administration REST APIs, Ingest REST APIs, and Query REST APIs in Telemetry Streaming.

> **See Also**
>
> Enabling Telemetry Streaming in Oracle AI Database for more information about the ORDS install script and PL/SQL packages

Complete the following steps to install Telemetry Streaming with a simple workspace on a ORDS setup having ingest, query and workspace admin users.

To enable ORDS on a workspace, you must first get an ORDS instance up and running having an ORDS DBA (say `ORDS_DBA`). Post that, complete the following steps.

1. Run as SYS `@$ORACLE_HOME/rdbms/admin/telemetry_install_ords.sql` (This creates ORDS-related packages in Telemetry Streaming).
2. Connect as `ORDS_DBA` and enable the ORDS schema for the TELEMETRY_DBA (Telemetry Administrator) and workspace (`WKSP1`). Run the following code blocks.

   a.
   ```
   exec ORDS.enable_schema(
       p_enabled             => TRUE,
       p_schema              => 'TELEMETRY_DBA',
       p_url_mapping_type    => 'BASE_PATH',
       p_url_mapping_pattern => 'TELEMETRY_DBA',
       p_auto_rest_auth      => FALSE
     );
   ```

   b.
   ```
   exec ORDS.enable_schema(
       p_enabled             => TRUE,
       p_schema              => 'TM$WKSP1',
   ```

```
    p_url_mapping_type    => 'BASE_PATH',
    p_url_mapping_pattern => 'wksp1',
    p_auto_rest_auth      => FALSE
  );
```

3. To set up the ORDS handlers for a workspace, connect to workspace admin (`UC1A`) and enable the Telemetry Streaming ORDS handlers. Run the following statement.

   ```
   exec dbms_telemetry_admin.enable_workspace_ords('WKSP1');
   ```

4. While connected as the workspace Admin, get the workspace ORDS Admin credentials. Run the following statement.

   ```
   select dbms_telemetry_admin.get_workspace_ords_admin_auth('WKSP1');
   ```

See Also

Managing ORDS Workspace Users and Using REST API for Workspace Administration for more information about managing ORDS workspace users and using REST API for workspace administration

5

Using Telemetry Streaming

This chapter explains how to use Telemetry Streaming to ingest and query data.

Topics:

- Ingesting Metric Data
  This section explains the different ways you can ingest metric data into Telemetry Streaming.
- Querying Metric Data
  This section explains the different ways you can query metric data in Telemetry Streaming.

5.1 Ingesting Metric Data

This section explains the different ways you can ingest metric data into Telemetry Streaming.

For clients outside of Oracle AI Database, Telemetry Streaming integrates with ORDS to allow REST API calls through ORDS endpoints to ingest data into Telemetry Streaming. For SQL clients, Telemetry Streaming provides PL/SQL packages to ingest data.

- Using PL/SQL to Ingest Metric Data
- Using REST APIs with ORDS to Ingest Metric Data

5.1.1 Using PL/SQL to Ingest Metric Data

Any existing Oracle user can be enabled as an ingest user for a workspace. However, a user can be added as an ingest user for only one workspace.

The `DBMS_TELEMETRY_INGEST` PL/SQL package enables an ingest user to ingest data into Telemetry Streaming either as a single sample ingest data or as `CLOB` data.

Table 5-1 `DBMS_TELEMETRY_INGEST` Package

Function	Description
`INGEST_METRICS`	For single metric sample ingestion
`INGEST_METRICS`	For metric ingestion as `CLOB`

See Also

DBMS_TELEMETRY_INGEST in *Oracle AI Database PL/SQL Packages and Types Reference* for more information about the `DBMS_TELEMETRY_INGEST` PL/SQL package.

5.1.2 Using REST APIs with ORDS to Ingest Metric Data

If you want an external client to push metric data through ORDS into Telemetry Streaming, you can use the REST APIs through ORDS endpoints. ORDS employs the OAuth2 authorization mechanism to ingest data using ORDS endpoints. Therefore, you must authenticate the requests using the client credentials (Client ID and Client Secret pair), which is created by the Telemetry ORDS administrator or the workspace administrator.

The client credentials flow in OAuth2 is a two-legged process. You use the client credentials to return an access token, which is then used to authenticate the API calls to the ORDS endpoints and ingest data.

To secure the ORDS APIs, Telemetry Streaming ensures that:

- All the REST APIs are authenticated using the OAuth2 Client Credentials protocol.
- A workspace administration user (who is an existing database user) can get the OAuth credentials (`client-id, client-secret`) for the ORDS administration user.

  Note

  A workspace administration user should enable ORDS using `DBMS_TELEMETRY_ADMIN.ENABLE_WORKSPACE_ORDS` before fetching the ORDS admin credentials.

- Only an ORDS user with administrative privileges can create the ORDS ingest and ORDS query users.

The following section provides the REST API documentation for the ingest REST APIs.

Note

- Ensure that ORDS is installed and Telemetry Streaming is REST enabled before using REST APIs.
- To make ORDS Ingest API calls, you need an OAuth2 access token to use for authorization. The token can be derived using the token URL by providing the Client ID and Client Secret of the ORDS ingest user.

See Also

Installing Oracle REST Data Services for more information about the ORDS installation.

Ingesting Data Using REST API

Once an ORDS ingest user is created using the ORDS admin user, it can be used to ingest data through the following API endpoints.

Table 5-2 REST API Summary

API Type	Summary
POST `/ords/<Workspace Name>/<Ords Ingest User>/ingest`	Ingests data into Telemetry Streaming using the ORDS API endpoint (Telemetry protocol)
POST `/ords/<Workspace Name>/<Ords Ingest User>/ingestlp`	Ingests data into Telemetry Streaming using the ORDS API endpoint (Line protocol)

Ingest Data (Telemetry protocol)

METHOD: **POST**

PATH: `/ords/<Workspace Name>/<Ords Ingest User>/ingest`

USER: ORDS Ingest User

SUMMARY: Ingests data into Telemetry Streaming using the ORDS API endpoint (Telemetry protocol)

DATA:

```
{ "metrics" : [
                 [<metric_name>, <tags as JSON>, <value>, <time in
secs since epoch>],
                 [<metric_name>, <tags as JSON>, <value>, <time in
secs since epoch>]
                 ..
              ]
}
```

HTTP CODE:

201 - Successfully inserted

```
{"metrics_data_size":<num>,"metrics_ingested":<num>,

"metrics_given":<num>,"metrics_metadata":<num>,"ingest_duration_ms":
<time in ms>,

"ingest_format":<format>,"ingest_method":<method>,"ingest_user_name"
:<ingest user name>}
```

400 - Unsuccessful
401 - Unauthorized

Ingest Data (Telemetry protocol)	

EXAMPLE:	PATH: `/ords/workspace1/ingest1/ingest`
	DATA:
	`{ "metrics" : [ ["scrape_duration_seconds", {"http_scheme":"http","net_host_port":"2112"}, 0, 1704700084], ["scrape_samples_scraped", {"http_scheme":"http","net_host_port":"2112"}, 12, 1704700085] .. ] }`
	CODE: 201 - Successfully inserted
	`{"metrics_data_size":611,"metrics_ingested":2, "metrics_given":2,"metrics_metadata":2,"ingest_duration_ms":110, "ingest_format":"telemetry","ingest_method":"ords","ingest_user_name":"INGEST1"}`

```
{ "metrics" : [
                 ["scrape_duration_seconds",
{"http_scheme":"http","net_host_port":"2112"}, 0, 1704700084],
                 ["scrape_samples_scraped",
{"http_scheme":"http","net_host_port":"2112"}, 12, 1704700085]
                 ..
              ]
}
```

Ingest Data (Line protocol)	
METHOD:	**POST**
PATH:	`/ords/<Workspace Name>/<Ords Ingest User>/ingestlp`
USER:	ORDS Ingest User
SUMMARY:	Ingests data into Telemetry Streaming using the ORDS API endpoint (Line protocol)
DATA:	`<measurement>[,<tag_key>=<tag_value>[,<tag_key>=<tag_value>]] <field_key>=<field_value>[,<field_key>=<field_value>] [<timestamp>]`
HTTP CODE:	201 - Successfully inserted `{"metrics_data_size":<num>,"metrics_ingested":<num>, "metrics_given":<num>,"metrics_metadata":<num>,"ingest_duration_ms":<time in ms>, "ingest_format":<format>,"ingest_method":<method>,"ingest_user_name":<ingest user name>}` 400 - Unsuccessful 401 - Unauthorized

Ingest Data (Line protocol)	

EXAMPLE:	PATH: `/ords/workspace1/ingest1/ingestlp`
	DATA:
	`trig,host=phoenix92613 sine=0 1752215852000000000`
	`mem,host=phoenix92613 sine=0 1752215852000000000`
	CODE: 201 - Successfully inserted
	`{"metrics_data_size":611,"metrics_ingested":2,`
	` "metrics_given":2,"metrics_metadata":2,"ingest_duration_ms":110,`
	`"ingest_format":"line","ingest_method":"ords","ingest_user_name":"IN`
	`GEST1"}`

Ingest Example Using cURL

1. Add a new ingest user.
   Use the following cURL command to add a new user named `INGEST1` with the role `INGEST` using workspace ORDS admin access token:

   Request:

   ```
   curl -k --request POST -i \
   -H "Authorization: Bearer {admin user token}" \
   -H "Content-Type: application/json" \
   --data '{ "user_name": "<USERNAME>", "role": "<ROLE>" }' \
   -v <ADD_USER_URL>
   ```

   Response:

   ```
   {"client_id":<client_id>, "client_secret":<client_secret key>}
   ```

   Example:

   ```
   curl -k --request POST -i \
   -H "Authorization: Bearer moVgM6Vi6TqJB0xbYezicA" \
   -H "Content-Type: application/json" \
   --data '{ "user_name": "INGEST1", "role": "INGEST" }' \
   -v http://example.com:8085/ords/wksp1/add_user
   ```

   Response:

   ```
   {"client_id":"aBcDefg1hij2k3l..","client_secret":"mNoPq45rst6u7xyz.."}
   ```

2. Get the access token using the client ID:client secret.

Request:

```
curl -k --request POST  \
    --user $INGEST_USER_CLIENT_ID:$INGEST_USER_CLIENT_SECRET \
    --data "grant_type=client_credentials" <TOKEN_URL>
```

Response:

```
{"access_token":<token>, "token_type":<token type>, "expires_in":<time in
seconds>}
```

Example:

```
curl -k --request POST  \
    --user aBcDefg1hij2k3l..:mNoPq45rst6u7xyz.. \
    --data "grant_type=client_credentials" <http://example.com:8085/ords/
wksp1/oauth/token>
```

Response:

```
{"access_token":"B1o38ikZ55tNMXsuPlksDQ","token_type":"bearer","expires_in"
:3600}
```

3. Ingest the data.
   Request:

```
curl -k --request POST -i -H "Authorization: Bearer {ingest user token}" \
    --header "Content-Type: application/json" \
    --data @<file_name> -v <INGESTION URL>
```

Response:

```
{"metrics_data_size":<num>,"metrics_ingested":<num>,

"metrics_given":<num>,"metrics_metadata":<num>,"ingest_duration_ms":<time
in ms>,

"ingest_format":<format>,"ingest_method":<method>,"ingest_user_name":<inges
t user name>}
```

Example (using the default **Telemetry protocol**):

```
curl -k --request POST -i -H "Authorization: Bearer
B1o38ikZ55tNMXsuPlksDQ" \
    --header "Content-Type: application/json" \
    --data @data.json -v https://example.com:8000/ords/workspace1/ingest1/
ingest
```

Response:

```
{"metrics_data_size":79,"metrics_ingested":1,
          "metrics_given":1,"metrics_metadata":1,"ingest_duration_ms":10,
"ingest_format":"telemetry","ingest_method":"ords","ingest_user_name":"inge
st1"}
```

Example (using **Line protocol**):

```
curl -k --request POST -i -H "Authorization: Bearer
B1o38ikZ55tNMXsuPlksDQ" \
    --header "Content-Type: application/json" \
    --data @data.json -v https://example.com:8000/ords/workspace1/ingest1/
ingestlp
```

Response:

```
{"metrics_data_size":79,"metrics_ingested":1,
         "metrics_given":1,"metrics_metadata":1,"ingest_duration_ms":10,

"ingest_format":"line","ingest_method":"ords","ingest_user_name":"ingest1"}
```

5.2 Querying Metric Data

This section explains the different ways you can query metric data in Telemetry Streaming.

You can use PromQL or SQL to query the time series data that is ingested in Telemetry Streaming.

- Using PromQL to Query Metric Data
  Learn about the supported PromQL queries and how to use them to query in Telemetry Streaming.
- Using REST API with ORDS for Querying
  This section describes the REST APIs used for querying.
- Using SQL to Query Metric Data
  Learn how you can use SQL for querying in Telemetry Streaming.

5.2.1 Using PromQL to Query Metric Data

Learn about the supported PromQL queries and how to use them to query in Telemetry Streaming.

To query the data using PromQL, you can use either of the following:

- PL/SQL APIs using the `DBMS_TELEMETRY_QUERY` package

  See Also

  DBMS_TELEMETRY_QUERY in *Oracle AI Database PL/SQL Packages and Types Reference* for more information about the `DBMS_TELEMETRY_QUERY` PL/SQL package.

- ORDS REST endpoints
  Querying using REST API is explained later in this section.

In Oracle AI Database 26ai, Release 23.26.0, only a subset of PromQL constructs are supported.

- Supported PromQL Queries for DBMS_TELEMETRY_QUERY
- Using Supported PromQL Query Operators

See Also

PromQL Querying Basics to get started with building queries using PromQL

5.2.1.1 Supported PromQL Queries for `DBMS_TELEMETRY_QUERY`

The following types of PromQL queries are supported in `DBMS_TELEMETRY_QUERY` for each query category.

Range/Instant PromQL Queries

- Vector Selector

  ```
  node_cpu_seconds_total{cpu="1"}
  ```

- Functions (rate/irate)

  ```
  rate(node_cpu_seconds_total{cpu="1"}[3m])
  ```

- Aggregate: sum, avg, count, max, min, stddev, stdvar, group, quantile, topk, bottomk, count_values

  ```
  sum(node_cpu_seconds_total)

  min(node_boot_time_seconds{device!="eth0"})

  stddev(node_disk_info{major!="11"})

  group(container_spec_cpu_period{job="cadvisor"})

  quantile(0.95,node_arp_entries{device=~"eth0"})

  topk(5,node_boot_time_seconds{job="node"})

  count_values("values",node_cpu_online)
  ```

- Aggregate by: sum, avg, count, max, min, stddev, stdvar, group, quantile, topk, bottomk, count_values

  ```
  avg by(cpu)(node_cpu_online{job=~"node"})

  count(go_gc_gogc_percent{job="node"})

  stdvar by(bios_date)(node_disk_info{job="node"})

  group by(address)(container_spec_cpu_period{job="cadvisor"})
  ```

```
quantile by(adminstate)(0.80,go_gc_gogc_percent{job="node"})

bottomk by(tags)(node_boot_time_seconds{job=~"node"})

count_values by(tags)("cpu",node_cpu_load)
```

Label Queries

- Fetch all the distinct keys in the tags column or for a given tag key, fetch all the distinct values of that corresponding key.

Series Queries

- Fetch a range of time series that matches a specific criteria, such as a label filter or metric name.

  ```
  node_cpu_seconds_total{cpu="1"}
  ```

See Also

PromQL Querying for more information about PromQL querying

5.2.1.2 Using Supported PromQL Query Operators

The following shows the usage of the supported PromQL queries for `DBMS_TELEMETRY_QUERY` in the initial version.

Range Queries

Range queries in PromQL are queries that span over a time range.

The format is as follows:

```
select dbms_telemetry_query.promql_range(promql_query, start_time_epoch,
end_time_epoch, step_size_seconds, fetch_sql[optional]) from dual;
```

Note

The `fetch_sql` value can be either 0 or 1. If it is 1, the SQL conversion of the PromQL query is returned; if it is 0 [default value], the result after executing the SQL is returned as JSON.

The range queries can be of 4 types:

- Simple expression queries

  ```
  select
  dbms_telemetry_query.promql_range('node_cpu_load{cpu="1"}',1389312400,13893
  12600,10) from dual;
  ```

- Simple aggregate queries (sum, avg, count, count_values, max, min, std deviation, std variance, group, quantile, top_k, bottom_k)

  ```
  select
  dbms_telemetry_query.promql_range('sum(node_cpu_load{cpu="1"})',1389312400,
  1389312600,10) from dual;
  ```

- Simple aggregate-by queries (sum, avg, count, count_values, max, min, std deviation, std variance, group, quantile, top_k, bottom_k)

  ```
  select dbms_telemetry_query.promql_range('sum by(tag)
  (node_cpu_load{cpu="1"})',1389312400,1389312600,10) from dual;
  ```

- Functions (Rate/IRate)

  ```
  select dbms_telemetry_query.promql_range('irate(node_cpu_load{cpu="1"}
  [5m])',1389312400,1389312600,10) from dual;
  ```

Instant Queries

Instant queries in PromQL request time series data for a particular timestamp.

The format is as follows:

```
select dbms_telemetry_query.promql_instant(promql_query, instant_time_epoch,
fetch_sql[optional]) from dual;
```

Note

The `fetch_sql` value can be either 0 or 1. If it is 1, the SQL conversion of the PromQL query is returned; if it is 0 [default value], the result after executing the SQL is returned as JSON.

Instant queries can be of four types.

- Simple expression queries

  ```
  select
  dbms_telemetry_query.promql_instant('node_cpu_load{cpu="1"}',1389312600)
  from dual;
  ```

- Simple aggregate queries (sum, avg, count, count_values, max, min, std deviation, std variance, group, quantile, top_k, bottom_k)

  ```
  select
  dbms_telemetry_query.promql_instant('sum(node_cpu_load{cpu="1"})',138931260
  0) from dual;
  ```

- Simple aggregate-by queries (sum, avg, count, count_values, max, min, std deviation, std variance, group, quantile, top_k, bottom_k)

  ```
  select dbms_telemetry_query.promql_instant('sum by(tag)
  (node_cpu_load{cpu="1"})',1389312600) from dual;
  ```

- Functions (Rate/IRate)

  ```
  select dbms_telemetry_query.promql_instant('rate(node_cpu_load{cpu="1"}
  [3m])',1389312600) from dual;
  ```

Label Queries

Label queries in PromQL fetch distinct keys or distinct values.

The format is as follows:

```
select dbms_telemetry_query.promql_label(promql_query, start_time_epoch,
end_time_epoch, fetch_sql[optional]) from dual;
```

> **Note**
>
> The `fetch_sql` value can be either 0 or 1. If it is 1, the SQL conversion of the PromQL query is returned; if it is 0 [default value], the result after executing the SQL is returned as JSON.

Label queries are of two types.

- Fetches all the distinct keys in the tags (if the promql query is empty)

  ```
  select dbms_telemetry_query.promql_label(' ',1389312400,1389312600) from
  dual;
  ```

- Fetches all the distinct values that a particular key can take in the tags

  ```
  select dbms_telemetry_query.promql_label('cpu',1389312400,1389312600) from
  dual;
  ```

Series Queries

Series queries in PromQL fetch the metadata (metric name and tags) of the time series that match the given PromQL query.

The format is as follows:

```
select dbms_telemetry_query.promql_series(promql_query, start_time_epoch,
end_time_epoch, fetch_sql[optional]) from dual;
```

> **Note**
>
> The `fetch_sql` value can be either 0 or 1. If it is 1, the SQL conversion of the PromQL query is returned; if it is 0 [default value], the result after executing the SQL is returned as JSON.

```
select
dbms_telemetry_query.promql_series('node_cpu_load{cpu="1"}',1389312400,1389312
600) from dual;
```

> **See Also**
>
> PromQL Querying for more information about PromQL querying

5.2.2 Using REST API with ORDS for Querying

This section describes the REST APIs used for querying.

Once an ORDS query user is created using the ORDS workspace administrator, the user can query data or view query statistics through the following REST API endpoints.

> **Note**
>
> Ensure that ORDS is installed and Telemetry Streaming is REST enabled before using REST APIs.

> **See Also**
>
> Installing Oracle REST Data Services for more information about the ORDS installation.

Table 5-3 REST API Summary

API Type	Summary
POST `/ords/<Workspace Name>/<Ords Query User>/<Query Type>`	Query Telemetry Streaming data using ORDS API endpoint

Query Data	
METHOD:	**POST**
PATH:	`/ords/<Workspace Name>/<Ords Query User>/<Query Type>`
QUERY TYPE:	`promql_range`, `promql_label`, `promql_series`, `promql_instant`
USER:	ORDS Query User
SUMMARY:	Query Telemetry Streaming data using ORDS API endpoint

Range Query	

DATA:	
	```
	{
	   "promql_query" :  <Promql_query>,
	   "start_time"   :  <start time in epoch>,
	   "end_time"     :  <end time in epoch>,
	   "step_size"    :  <step size>
	}
	```

HTTP CODE:	201 - Successfully executed
	```
	{
	   "status": "success",
	   "data": {
	     "resultType" : "matrix",
	     "result":
	     [
	       {
	         "metric":
	         {
	           "__name__": <metric_name>,
	           <tag_key>: <tag_value>
	         },
	         "value":
	          [
	           <epoch_time>,
	           <metric_value>
	          ]
	       }
	     ]
	   }
	}
	```
	400 - Unsuccessful
	401 - Unauthorized

Range Query

EXAMPLE

PATH: `/ords/workspace1/query1/promql_range`

DATA:

```
{
     "promql_query" :  "node_cpu_load{tag="3"}",
     "start_time"   :  1389312580,
     "end_time"     :  1389312600,
     "step_size"    :  20
}
```

CODE: 201 - Successfully executed

```
{
  "status": "success",
  "data": {
    "resultType": "matrix",
    "result": [
      {
        "metric": {
          "tag": "3"
        },
        "values": [
          [
            1389312580,
            "10.000000"
          ],
          [
            1389312600,
            "10.000000"
          ]
        ]
      }
    ]
  }
}
```

Instant Query

DATA:

```
{
     "promql_query" :  <Promql_query>,
     "point_time"   :  <time in epoch>
}
```

Instant Query	
HTTP CODE:	201 - Successfully executed `{ "status": "success", "data": { "resultType" : "vector", "result": [ { "metric": { "__name__": <metric_name>, <tag_key>: <tag_value> }, "value": [ <epoch_time>, <metric_value> ] } ] } }` 400 - Unsuccessful 401 - Unauthorized

```
{
    "status": "success",
    "data": {
      "resultType" : "vector",
      "result":
      [
        {
          "metric":
          {
            "__name__": <metric_name>,
            <tag_key>: <tag_value>
          },
          "value":
            [
              <epoch_time>,
              <metric_value>
            ]
          }
      ]
   }
}
```

Instant Query	
EXAMPLE	PATH: `/ords/workspace1/query1/promql_point` DATA:

```
{
   "promql_query" :  "node_cpu_load{tag="3"}",
   "point_time"   :  1389312580,
}
```

CODE: 201 - Successfully executed

```
 {
  "status": "success",
  "data": {
    "resultType": "vector",
    "result": [
      {
        "metric": {
          "tag": "3"
        },
        "values": [
          [
            1389312580,
            "10.000000"
          ]
        ]
      }
    ]
  }
}
```

Series Query

DATA:

```
{
    "promql_query" :  <Promql_query>,
    "start_time"   :  <start time in epoch>,
    "end_time"     :  <end time in epoch>,
}
```

Series Query	

HTTP CODE:	201 - Successfully executed
	`{`
	`  "status" : "success",`
	`  "data" : [`
	`  {`
	`      "__name__": <metric_name>,`
	`      <tag_key>:  <tag_value>`
	`   }`
	` ]`
	`}`
	400 - Unsuccessful
	401 - Unauthorized

EXAMPLE	PATH: `/ords/workspace1/query1/promql_series`
	DATA:
	`{`
	`  "promql_query" :  "node_cpu_load{tag="3"}",`
	`  "start_time"   :  1389312580,`
	`  "end_time"     :  1389312600,`
	`}`
	CODE: 201 - Successfully executed
	`{`
	`  "status" : "success",`
	`  "data" : [`
	`  {`
	`    "__name__" : "node_cpu_load",`
	`    "tag" : "3"`
	`  }`
	`  ]`
	`}`

Label Query	

DATA:	`{`
	`    "promql_query" :  <Promql_query>,`
	`    "start_time"   :  <start time in epoch>,`
	`    "end_time"     :  <end time in epoch>,`
	`}`

Label Query	

HTTP CODE:	201 - Successfully executed
	`{`
	`    "status" : "success",`
	`    "data" : [`
	`         <tag_value>`
	`    ]`
	`}`
	400 - Unsuccessful
	401 - Unauthorized

EXAMPLE	PATH: `/ords/workspace1/query1/promql_label`
	DATA:
	`{`
	`   "promql_query" :  "tag",`
	`   "start_time"   :  1389312580,`
	`   "end_time"     :  1389312600,`
	`}`
	CODE: 201 - Successfully executed
	`{`
	`     "status" : "success",`
	`     "data" : [`
	`       "3"`
	`     ]`
	`}`

Query Example Using cURL

1. Add a new query user.
   Request:

   ```
   curl -k --request POST -i \
    -H "Authorization: Bearer {admin user token}" \
    -H "Content-Type: application/json" \
    --data '{ "user_name": "<USERNAME>", "role": "<ROLE>" }' \
    -v <ADD_USER_URL>
   ```

   Response:

   ```
    {"client_id":<client_id>, "client_secret":<client_secret key>}
   ```

   Example:

   ```
    curl -k --request POST -i \
    -H "Authorization: Bearer moVgM6Vi6TqJB0xbYezicA" \
   ```

```
 -H "Content-Type: application/json" \
 --data '{ "user_name": "QUERY1", "role": "QUERY" }' \
 -v http://example.com:8085/ords/wksp1/add_user
```

Response:

```
{"client_id":"fymGQ8uO8zuPTiBWrw8CEw..","client_secret":"fPJ3RcJlR_XEmlrC27
6dEg.."}
```

2. Generate an access token for the query user.
   Generate an OAuth token using the `client_id` and `client_secret`.

   Request:

```
 curl -k --request POST  \
    --user $QUERY_USER_CLIENT_ID:$QUERY_USER_CLIENT_SECRET \
    --data "grant_type=client_credentials" <TOKEN_URL>
```

   Response:

```
{"access_token":<token>, "token_type":<token type>, "expires_in":<time in
seconds>}
```

   Example:

```
 curl -X POST \
 --user fymGQ8uO8zuPTiBWrw8CEw..:fPJ3RcJlR_XEmlrC276dEg.. \
 --data "grant_type=client_credentials" \
 http://example.com:8085/ords/wksp1/oauth/token
```

   Response:

```
{"access_token":"GVCvNXcqRG9OBZM98zHPcQ","token_type":"bearer","expires_in"
:3600}
```

3. Run the PromQL Range/PromQL Instant/PromQL Series/PromQL Label query using the access token.
   Use the generated token (for example, `GVCvNXcqRG9OBZM98zHPcQ`) to query the PromQL endpoint as the `QUERY1` user.

   The following examples demonstrate the query requests and responses for each query type.

   PromQL Range:

   Example:

```
curl -k --request POST -i -H "Authorization: Bearer
GVCvNXcqRG9OBZM98zHPcQ" \
   --header "Content-Type: application/json" \
   --data '{ "promql_query": "met1",     "start_time":  1759752400,
"end_time":  1759752700, "step_size": 10}'  \
   -v http://example.com:8085/ords/wksp1/QUERY1/promql_range
```

Response:

```
{
  "status" : "success",
  "data" :
  {
    "resultType" : "matrix",
    "result" :
    [
      {
        "metric" :
        {
          "__name__" : "met1",
          "tag1" : "val1"
        },
        "values" :
        [
          [
            1759752590,
            "10.000000"
          ]
        ]
      }
    ]
  }
}
```

PromQL Instant

Example:

```
curl -k --request POST -i -H "Authorization: Bearer
GVCvNXcqRG9OBZM98zHPcQ" \
   --header "Content-Type: application/json" \
   --data '{ "promql_query": "met1", "point_time" : 1751460370}'  \
   -v http://example.com:8085/ords/wksp1/QUERY1/promql_point
```

Response:

```
{
  "status" : "success",
  "data" :
  {
    "resultType" : "vector",
    "result" :
    [
      {
        "metric" :
        {
          "__name__" : "met1",
          "tag1" : "val1"
        },
        "value" :
        [
          1759752700,
          "10.000000"
        ]
      }
    ]
  }
}
```

PromQL Series

Example:

```
curl -k --request POST -i -H "Authorization: Bearer 
GVCvNXcqRG9OBZM98zHPcQ" \
   --header "Content-Type: application/json" \
   --data '{ "promql_query": "met1",     "start_time":  1759752400,
"end_time":  1759752700}'  \
   -v http://example.com:8085/ords/wksp1/QUERY1/promql_series
```

Response:

```
{
  "status" : "success",
  "data" :
  [
    {
      "__name__" : "met1",
      "tag1" : "val1"
    }
  ]
}
```

PromQL Label

Example:

```
curl -k --request POST -i -H "Authorization: Bearer 
GVCvNXcqRG9OBZM98zHPcQ" \
   --header "Content-Type: application/json" \
   --data '{ "promql_query": "tag1",     "start_time":  1759752400,
"end_time":  1759752700}'  \
   -v http://example.com:8085/ords/wksp1/QUERY1/promql_label
```

Response:

```
{
  "status" : "success",
  "data" :
  [
    "val1"
  ]
}
```

5.2.3 Using SQL to Query Metric Data

Learn how you can use SQL for querying in Telemetry Streaming.

Telemetry Streaming is built on Oracle AI Database, and so it supports SQL to query the time series data. The advantage of using SQL is its ability to construct complex queries that can give better insight into the data.

A workspace query user can access metrics data using SQL on the `TM$<WORKSPACE_NAME>.TELEMETRY_METRICS` view of the workspace.

For Example:

```
select metric_name, metric_tags, metric_value, metric_time from
TM$WORKSPACE1.TELEMETRY_METRICS.
```

See Also

Table 8 in Tables and Views for more information about the workspace administration views

6

Errors and Troubleshooting

This chapter discusses the likely errors that can arise when using Telemetry Streaming and how to fix them.

The following sections provide an overview of the potential errors and the troubleshooting steps required to resolve them.

Workspace Management Errors

Error	Example	Troubleshooting

ORA-00959: tablespace 'TBS' does not exist	`SQL> exec dbms_telemetry_workspace.create_workspace('workspace1','tbs');`
	`BEGIN dbms_telemetry_workspace.create_workspace('workspace1','tbs'); END;`
	
	`*`
	`ERROR at line 1:`
	`ORA-00959: tablespace 'TBS' does not exist`
		Tablespace for the workspace should be created before creating the workspace.

ORA-20004: "ADMIN1":user does not exist	`SQL> exec dbms_telemetry_workspace.enable_workspace_admin('workspace1','admin1');`
	`BEGIN dbms_telemetry_workspace.enable_workspace_admin('workspace1','admin1'); END;`
	
	`*`
	`ERROR at line 1:`
	`ORA-20004: "ADMIN1":user does not exist`
		Admin user should be created before enabling for the workspace. It is not created by the PL/SQL API.
		Resolution: Create a DB user 'admin1' with connect privilege and retry the operation.

Error	Example	Troubleshooting

ORA-20012: "WORKSPACE1": has users. use force=>true	`SQL> exec dbms_telemetry_workspace.drop_workspace('workspace1');`
	`BEGIN dbms_telemetry_workspace.drop_workspace('workspace1'); END;`
	
	`*`
	`ERROR at line 1:`
	`ORA-20012: "WORKSPACE1": has users. use force=>true`
		Workspaces with active users cannot be dropped without 'force' option being 'true'.

Workspace Administration Errors

Error	Example	Troubleshooting

ORA-44001: invalid schema	`SQL> exec dbms_telemetry_admin.enable_workspace_user('workspace1','user1','INGEST');`
	`BEGIN dbms_telemetry_admin.enable_workspace_user('workspace1','user1','INGEST'); END;`
	
	`*`
	`ERROR at line 1:`
	`ORA-44001: invalid schema`
		Ingest or query user should be created before enabling for the workspace.
		Resolution: Create a DB user 'user1' with connect privilege and retry the operation.

ORA-20008: inges: invalid role	`SQL> exec dbms_telemetry_admin.enable_workspace_user('workspace1','user2','INGES');`
	`BEGIN dbms_telemetry_admin.enable_workspace_user('workspace1','user2','INGES'); END;`
	
	`*`
	`ERROR at line 1:`
	`ORA-20008: inges: invalid role`
		The role can be only one of 'ingest', 'query' or 'ingest|query'.
		Resolution: Correct the role to a valid value.

Error	Example	Troubleshooting
ORA-20010: "USER1": already added to workspace "WORKSPACE1"	`SQL> exec dbms_telemetry_admin.enable_workspace_user('workspace1','user1','INGEST'); BEGIN dbms_telemetry_admin.enable_workspace_user('workspace1','user1','INGEST'); END;` `*` `ERROR at line 1:` `ORA-20010: "USER1": already added to workspace "WORKSPACE1"`	User can be added to only one of the workspaces and only once. **Note** For changing the role of an already assigned user, disable the user from the workspace and re-enable with new role.

Ingest Errors

Issue	Example	Troubleshooting

Metrics not ingested because the time epoch is not in the last one hour. Older metrics are discarded.	
	```
	begin

	dbms_output.put_line(dbms_telemetr
	y_ingest.ingest_metrics(

	'speed',

	'{"type":"car","unit":"mph"}',

	100,

	1000));
	end;
	/


	{"metrics_data_size":60,"metrics_i
	ngested":0,"metrics_given":1,"metr
	ics_metadata
	":0,"ingest_duration_ms":10,"inges
	t_format":"telemetry","ingest_meth
	od":"plsql
	","ingest_user_name":"\"USER1\""}
	```
		Issue: `"metric_ingested":0` implies that no rows were ingested.s
		Resolution: The metrics time should be greater than 3600 seconds from the time of ingestion and less than equal to current time.

PromQL Errors

Error	Example	Troubleshooting

ORA-52002: PromQL error: start time and end time cannot be zero for range or point queries.	
	```
	SQL> select
	dbms_telemetry_query.promql_instan
	t('node_cpu_load{tag!~"1|2"}',0)
	from dual;
	   ERROR:
	   ORA-52002: PromQL error: start
	time and end time cannot be zero
	for range or point queries
	```
		Passing start time and end time as 0 for Instant/Range PromQL query.

Error	Example	Troubleshooting

ORA-00067: invalid value 0 for parameter step size; must be at least 1.	`SQL> select dbms_telemetry_query.promql_range('node_cpu_load{tag=~"8|3"}',1389312600 - 500, 1389312600,0) from dual;`
	`   ERROR:`
	`   ORA-00067: invalid value 0 for parameter step size; must be at least 1`
		Passing Step Size that is less than or equal to zero for range PromQL query.

ORA-52002: PromQL error: Aggregate functions are not used with Series Query.	`SQL> select dbms_telemetry_query.promql_series('avg(node_cpu_load{tag="3"})', 0, 0) from dual;`
	`   ERROR:`
	`   ORA-52002: PromQL error: Aggregate functions are not used with Series Query`
		Using aggregate or aggregate-by operator for label/series PromQL query.

ORA-52002: PromQL error: Unknown Expr Node type passed for Label query. Expr Node type 1.	`SQL> select dbms_telemetry_query.promql_label('avg(node_cpu_load{tag="3"})', 0, 0) from dual;`
	`   ERROR:`
	`   ORA-52002: PromQL error: Unknown Expr Node type passed for Label query. Expr Node type 1`
		Using aggregate or aggregate-by operator for label/series PromQL query.

ORA-52002: PromQL error: Incorrect result type passed as input	`SQL> select dbms_telemetry_query.promql_label('tags', 0, 0, 8) from dual;`
	`   ERROR:`
	`   ORA-52002: PromQL error: Incorrect result type passed as input`
		Passing invalid result_type values for PromQL query.

ORA-52002: PromQL error: Unknown Expr Node type passed for Label query. Expr Node type 5	`SQL> select dbms_telemetry_query.promql_label('irate(node_cpu_load{tag="3"}[8m])', 0, 0) from dual;`
	`   ERROR:`
	`   ORA-52002: PromQL error: Unknown Expr Node type passed for Label query. Expr Node type 5`
		Applying rate/irate functions in PromQL label or series-based queries.

Error	Example	Troubleshooting

ORA-52002: PromQL error: rate/ irate functions are not used with Series Query	
	```
	SQL> select
	dbms_telemetry_query.promql_serie
	s('rate(node_cpu_load{tag="3"}
	[8m])', 0, 0) from dual;
	   ERROR:
	   ORA-52002: PromQL error: rate/
	irate functions are not used with
	Series Query
	```
		Applying rate/irate functions in PromQL label or series-based queries.

A.1 Tables and Views

The following tables and views are used in Telemetry Streaming.

Oracle DBA Views

The following views are accessible to users having the Oracle DBA role.

Table 1 `TELEMETRY_DBA.TELEMETRY_WORKSPACES`

Columns	Data Type	Description
`WORKSPACE_ID`	NUMBER	Workspace ID
`WORKSPACE_NAME`	VARCHAR2(128)	Name of the workspace
`TABLESPACE_NAME`	VARCHAR2(128)	Default tablespace of the workspace
`CREATION_TIME`	DATE	Time at which this workspace was created

Table 2 `TELEMETRY_DBA.TELEMETRY_ADMINS`

Columns	Data Type	Description
`WORKSPACE_ID`	NUMBER	Workspace ID
`ADMIN_USER`	VARCHAR2(128)	The database user who is assigned as an ADMIN to the workspace ID
`ADMIN_USERID`	NUMBER	User ID of the `ADMIN_USER`
`CREATION_TIME`	DATE	Time at which the user was assigned as an admin user

Admin User Views

The following views are accessible to users with the workspace administrator role.

Table 3 `TELEMETRY_DBA.TELEMETRY_WORKSPACE_USERS`

Column	Data Type	Description
`WORKSPACE_ID`	NUMBER	This is an automatically generated unique ID for every workspace created
`WORKSPACE_NAME`	VARCHAR2(128)	Name of the workspace given at the time of creation
`USER_NAME`	VARCHAR2(128)	The database user name of the user associated with this workspace
`USER_ROLE`	VARCHAR2(128)	'INGEST', 'QUERY' or 'INGEST|QUERY' are the valid values

Table 4 `TM$<workspace name>.TELEMETRY_INGEST_STATS`

Column	Data Type	Description
`PAYLOAD_SIZE`	NUMBER	The size of the ingest payload in bytes
`METRICS_INGESTED`	NUMBER	Number of metrics ingested

Table 4 (Cont.) `TM$<workspace name>.TELEMETRY_INGEST_STATS`

Column	Data Type	Description
`METRICS_DISCARDED`	NUMBER	Number of metrics rejected
`INGEST_USERNAME`	VARCHAR2(128)	Username of the ingest user
`INGEST_DURATION_MILLISECONDS`	NUMBER	Time taken for ingest in ms
`INGEST_METHOD`	VARCHAR2(200)	(PLSQL or REST)
`INGEST_TIMESTAMP`	TIMESTAMP(6)	The time of ingestion

Table 5 `TM$<workspace name>.TELEMETRY_INGEST_DISCARDS`

Column	Data Type	Description
`METRIC_NAME`	VARCHAR2(512)	The metric name
`METRIC_TAGS`	VARCHAR2(4000)	The metric tags
`METRIC_VALUE`	NUMBER	The metric value
`METRIC_TIME_EPOCH`	NUMBER	Time in epoch
`INGEST_PAYLOAD_ARRIVAL_TIME`	NUMBER	Epoch of ingest payload receive
`DISCARD_REASON`	VARCHAR2(500)	Why was the ingest discarded?
`DISCARD_TIMESTAMP`	TIMESTAMP(6)	The time at which the ingest was discarded

Table 6 `TM$<workspace name>.TELEMETRY_QUERY_STATS`

Column	Data Type	Description
`QUERY_TYPE`	VARCHAR2(200)	(Instant, Range, Label, or Series)
`START_TIME_EPOCH`	NUMBER	The query start time in epoch
`END_TIME_EPOCH`	NUMBER	The query end time in epoch
`STEP_SIZE_SECONDS`	NUMBER	The Step Size for range queries
`QUERY_USERNAME`	VARCHAR2(128)	The username of the query user
`QUERY_DURATION_MILLISECONDS`	NUMBER	The time taken for the ingest in ms
`QUERY_METHOD`	VARCHAR2(200)	(PLSQL or REST)
`QUERY_TIMESTAMP`	TIMESTAMP(6)	The time of the query

Table 7 `TM$<workspace name>.TELEMETRY_ADMIN_LOG`

Column	Data Type	Description
`LOG_MESSAGE`	VARCHAR2(4000)	Message Text
`LOG_TIMESTAMP`	TIMESTAMP(6)	Th timestamp of the message

Note

Query stats are gathered and inserted into stats table through an autonomous transaction as a part of the query. As a result, there is a write operation into the database as a part of the query. This prevents PromQL queries to be executed from Read-only, standby databases.

Query User Views

The following views are accessible to users with the query user role.

Table 8 `TM$<workspace name>.TELEMETRY_METRICS`

Column	Data Type	Description
`METRIC_NAME`	VARCHAR2(512)	The metric name part of the time series
`METRIC_TAGS`	VARCHAR2(4000)	The tags part of the time series in JSON format
`METRIC_VALUE`	NUMBER	The point-in-time value of the time series
`METRIC_TIME_EPOCH`	NUMBER	The time in epoch

Index