Anomalo -> Data Catalog Integration

This integration synchronizes data quality status from Anomalo to your data catalog.

Questions? support@anomalo.com

The integration operates in batch mode. When run it collects the latest data quality status for each monitored table, locates that table in your catalog, and synchronizes quality metadata to the catalog.

Supported catalogs

- Databricks Unity Catalog
- Google Dataplex
- Microsoft Purview
- Collibra (coming soon)
- JSON export (coming soon)

How to run

The integration requires python 3. To configure:

Setup python environment

We recommend using `venv` to keep dependencies well-managed.

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

Configuration

The catalog integration reads Anomalo and catalog config and secrets from the environment. You can create a `.env` file containing the settings or set them in the environment directly.

You will need your Anomalo deployment's base url and api token.

```
# Replace with the hostname or base url of your Anomalo deployment
ANOMALO_INSTANCE_HOST="https://app.anomalo.com"

# Create an API key for your Anomalo deployment by navigating to Settings > Account > API Keys
ANOMALO_API_SECRET_TOKEN="<your api token>"

# If Anomalo is deployed as a Snowflake Native App,
#    use ANOMALO_AUTHORIZATION_HEADER instead of ANOMALO_API_SECRET_TOKEN
ANOMALO_AUTHORIZATION_HEADER="Snowflake Token=<short-lived oauth token>"
```

See below for instructions on catalog-specific configuration.

Running the integration

Once you've set up the config for your catalog, run the integration.

```
# List catalog targets
python anomalo-catalog.py --catalogs

# Sync table quality status to Google Dataplex
python anomalo-catalog.py --catalog dataplex

# Sync table quality status to Microsoft Purview
python anomalo-catalog.py --catalog purview
```

Runtime options

Run `python anomalo-catalog.py --help` to see available options.

Selecting a specific Anomalo organization

By default, the integration reads tables from the Anomalo organization last accessed by the owner of the API key.

To sync from a specific organization, find the org id using `--list-anomalo-organizations` and then specifiy it using `--anomalo-organization-id <ORG_ID>`.

Caution: Do not sync multiple orgnaizations concurrently, or change organization in the UI while running the catalog sync.

```
# List available organizations
python anomalo-catalog.py --list-anomalo-organizations

# Sync check result data from Anomalo organization 1 to Purview
python anomalo-catalog.py --catalog purview --anomalo-organization-id 1
```

Catalog-specific config

Databricks Unity Catalog

Set Databricks workspace hostname, SQL warehouse UID, and API access token in the environment.

```
DATABRICKS_HOSTNAME="dbc-012345.cloud.databricks.com"
# in Databricks web console > SQL Warehouses > click desired warehouse > UID is in the URL
#    e.g. UID is 98f7654321 in warehouse URL /sql/warehouses/98f7654321
DATABRICKS_WAREHOUSE_UID="98f7654321"
# in Databricks web console > User Settings > Access tokens
DATABRICKS_ACCESS_TOKEN="<access token>"
```

Notes

- This integration works on tables, not views, due to limitations on Databrick's COMMENT SQL.
- This integration will overwrite existing comments on the table
- The Anomalo name for your Databricks data sources must include a single dash or underscore followed by the Databricks catalog store name. e.g. `DB Nickname-main` or `Prod DBX_main` for the `main` catalog store

Google Dataplex

In Google Cloud IAM, create a service account and grant it `bigquery.tables.update` using the **BigQuery Data Editor** role `roles/bigquery.dataEditor` or a custom role.

Download the service account credentials json object and **save it to a file named `google-service-account-key.json` in this directory**.

If you want to write DQ summary data into Dataplex, the service account also needs these Dataplex permissions. The quickest way is to use the **Dataplex Administrator** and **Dataplex Catalog Admin** roles `roles/dataplex.admin`, `roles/datacatalog.admin`.

- `resourcemanager.projects.get`
- `dataplex.entries.list`
- `dataplex.entries.search`
- `dataplex.aspectTypes.create`
- `dataplex.aspectTypes.get`
- `dataplex.aspectTypes.update`
- `dataplex.aspects.update`
- `dataplex.aspects.attach`
- `dataplex.aspects.list`

You can change the integration's behavior with these command-line arguments:

- `--update-table-description` - write to the BigQuery table description (plain text content visible in both BigQuery and Dataplex)
- `--no-update-labels` - don't update anomalo-specific labels on the table
- `--no-update-aspect` - don't write to a custom aspect in Dataplex (rich text content visible only in Dataplex)

Microsoft Purview

Create a Microsoft Entra service principal (aka application) using these instructions

You will need your `Tenant Id` (search for `Tenant Properties` in the Azure portal), plus the `Application (client) Id` and the `client_secret`.

```
ENTRA_TENANT_ID="<Entra TENANT ID>"
ENTRA_CLIENT_ID="<Application CLIENT ID>"
ENTRA_CLIENT_SECRET="<Application CLIENT SECRET>"
# For Purview API root url, in Purview go to Settings > Account and use the
#    Azure resource name of your Purview account followed by .purview.azure.com
PURVIEW_ROOT_URL="PurviewResourceName.purview.azure.com"
```

When this script first runs, it registers a custom business metadata category named `AnomaloDQ`. Data Quality summary and a deep link to Anomalo are added to this category each time the script is run.

You can change the integration's behavior by adding these command-line arguments:

- `--no-update-labels` - don't update anomalo-specific labels on the table
- `--no-update-aspect` - don't update the content in the AnomaloDQ business metadata
- `--no-update-endorsement` - don't add or remove a PowerBI endorsement on tables based on check pass/fail status
- `--force-update-typedefs` - attempt to re-create the custom business metadata typedefs even if they already exist

JSON Export (coming soon)

No additional config needed

Collibra (coming soon)

```
COLLIBRA_HOSTNAME="<your collibra hostname>"
COLLIBRA_USER="<collibra username>"
COLLIBRA_PASSWORD="<collibra password>"
```

Configuring the Collibra UI

Add Anomalo data to Collibra UI

When this script first runs, it creates new characteristics in Collibra.

After you run this integration once, you need to manually map and place these newly created elements in the Collibra UI. These steps must be completed in the Collibra UI.

1. Navigate to Settings -> Operating Model -> Asset Types and click into Table (a system managed Asset Type under Data Asset -> Data Structure).
2. On the left, click into Global Assignment -> Characteristics -> Add a Characteristic -> Search "anomalo" and add the Summary and Relation created above.
3. Then click Edit Layout - the left panel should contain the newly added elements. Click and drag the "Anomalo Data Quality Summary" Attribute and the "Anomalo Data Quality Check" Relation onto the right panel into the desired section of the UI, preferably near the top so the data quality results will be visible without needing to scroll down very far.
4. Lastly, click Publish at the top to save the changes.

Each newly created Attribute must be assigned to the newly created "Anomalo Data Quality Check" Asset Type. Repeat the steps above, adding Category, Check Status, Check Result, Check Run URL, and Anomalo Sync Time to the layout.

Note - these steps must be repeated for each Asset Type in Collibra. If monitored assets are of an additional (e.g. "View") or custom Asset Type in Collibra then repeated this config setup for each Asset Type. An additional Relation Type will also need to be created for each Asset Type in use other than Table.

Customize check result table in Collibra UI

Lastly, an additional one-time step must be completed through the UI to get all of the check result attributes to populate into the Table (or other Asset Type) page.

1. Navigate (in Collibra) to any table being monitored by Anomalo. The "has Anomalo Data Quality Check" Relation should now be populated with all of the checks currently running in Anomalo.

2. Click “Edit Fields” in the top right of the grid, and then Select Fields from the bottom left to select and add the Attributes created above - Check Status, Check Result, Check Run URL, Anomalo Sync Time, and Category.
3. You can also rename any of the attributes (e.g. Name -> Check Name) as well as re-arrange the order of the columns in the grid as desired.

This process customizes the results grid for all assets of the same type. Repeat to costomize the results grid for other Asset Types (e.g. View) in use.

Deploying the integration as an Azure Function

To run the the catalog integration on a regular basis, we recommend hosting it in an Azure Function (or similar serverless environment).

Here are instructions for running in an Azure Function App:

In the Azure portal, create a new Function App

1. Create a new Function App (Portal home > Create a resource > Function App)
2. Choose Consumption or Flex Consumption (if you need virtual networking support to reach your Anomalo or Purview deployments)
3. Configure your function with these settings:
   - Basics
     - Choose your preferred Subscription, Resource Group, Function App name, and Region
     - Runtime stack: **Python**
     - Version: **3.12** or newer
     - Instance size: **2048 MB**
   - Networking
     - Enable public access: **On** so that we can access the advanced configuration ui
     - Enable virtual network integration: **On** if Anomalo is deployed in your virtual network
     - Virtual Network: if enabled, select the virtual network that has access to your Anomalo deployment
   - Deployment
     - Continuous deployment: **Disable**
4. Click **Create** when you’re satisfied with the function app settings, and wait for the **Your deployment is complete** message
5. Click **Go to resource**

Configure the integration

Navigate to Settings > Environment variables in your newly created Function App and add App settings:

- Add each of the config settings and secrets you gathered above - `ANOMALO_INSTANCE_HOST`, `ENTRA_TENANT_ID`, `PURVIEW_ROOT_URL`, etc
- Add `TIMER_SCHEDULE_CRON` with a value of `0 0 8 * * *` to run the integration daily at 8am UTC
- Add `CLI_ARGS` with all of the command line arguments you need passed to the catalog integration

Don’t forget to click **Apply** at the bottom of the App settings page after adding these variables.

```
# Example settings
TIMER_SCHEDULE_CRON="0 0 8 * * *"
CLI_ARGS="--catalog purview --anomalo-organization-id 1"
```

Create the integration function using the Azure CLI

We are going to use the Azure CLI, `az`, to create a function from the integration zip file.

```
az login

# Optionally set the active subscription if needed
az account set --subscription <azure_subscription_id>

# Run these in the directory containing this README.md file and anomalo-catalog.py

# Create a zip with the catalog files and Azure Function config
zip -r catalog-package.zip adapters AnomaloCatalogAzureTask anomalo_api.py \
```

```
    anomalo-catalog.py README.md host.json requirements.txt
# Deploy the zip to your Function App
az functionapp deployment source config-zip -g <YourResourceGroupName> \
    -n <YourFunctionAppName> --src catalog-package.zip

# Example output
#  Getting scm site credentials for zip deployment
#  Starting zip deployment. This operation can take a while to complete ...
#  Deployment endpoint responded with status code 202
#  Waiting for sync triggers...
#  Checking the health of the function app
#  "Deployment was successful."
```

In the Azure portal, refresh the Function App overview page and you will now see a function named AnomaloCatalogAzureTask.

Note If you update the Environment variables, you may need to re-deploy the zip to force a reload of your function's runtime environment with the new variables.

Enable logging

Navigate to the **AnomaloCatalogAzureTask** function and select the **Logs** tab.

If Application Insights is not configured, configure it here and turn on Application Insights to ensure the integration's logs are captured.

Test Azure Function

- Navigate to the **AnomaloCatalogAzureTask** function and select the **Code + Test** tab.
- Confirm that the Logs view is set to App Insights Logs and says `Connected`.
- Click **Test/Run** to execute the integration.
  - Under **Input**, ensure `_master` is selected for the Key
  - Click `Run`
  - In Output you should see HTTP resposne code of `202 Accepted`
- Wait and watch the Logs view to monitor execution

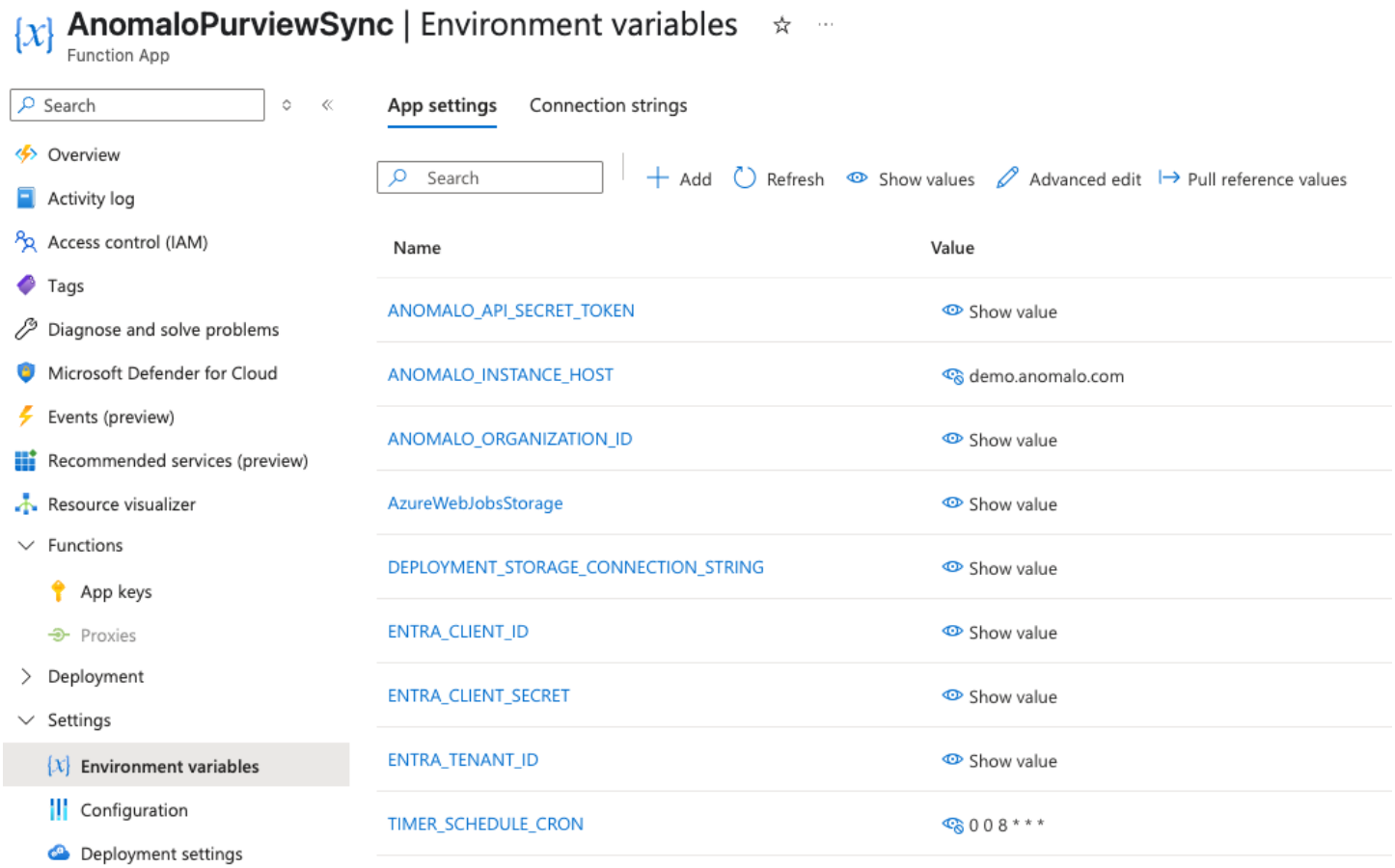

Figure 1: function config screenshot