Create Databricks Tables Connections

Contents:

You can create a connection to Azure Databricks tables from the Trifacta platform. This section describes how to create connections of this type.

- Databricks Tables provides a JDBC-based interface for reading and writing datasets in ADLS or WASB. Using the underlying JDBC connection, you can access your ADLS or WASB data like a relational datastore, run jobs against it, and write results back to the datastore as JDBC tables.
- Your connection to Databricks Tables leverages the SSO authentication that is native to Azure Databricks. For more information on Databricks Tables, see *https://docs.microsoft.com/en-us/azure/databricks/data/tables*.

Limitations

- Ad-hoc publishing of generated results to Databricks Tables is not supported.
- Creation of datasets with custom SQL is not supported.
- Integration with Kerberos or secure impersonation is not supported.
- Some table types and publishing actions are not supported. For more information, see *Using Databricks Tables*.

Pre-requisites

- The Trifacta platform must be installed on Azure and integrated with an Azure Databricks cluster.
  - See *Install for Azure*.
  - See *Configure for Azure Databricks*.

> **NOTE:** For job execution on Spark, the connection must use the Spark instance on the Azure Databricks cluster. No other Spark instance is supported. You can run jobs from this connection through the Photon running environment. For more information, see *Running Environment Options*.

- This connection interacts with Databricks Tables through the Hive metastore that has been installed in Azure Databricks.

> **NOTE:** External Hive metastores are not supported.

Insert Databricks Access Token

Each user must insert a Databricks Personal Access Token into the user profile. For more information, see *Databricks Personal Access Token Page*.

Enable

To enable Databricks Tables connections, please complete the following:

> **NOTE:** Typically, you need only one connection to Databricks Tables, although you can create multiple connections.

> **NOTE:** This connection is created with SSL automatically enabled.

Steps:

1. You can apply this change through the *Admin Settings Page* (recommended) or `trifacta-conf.json`

   . For more information, see *Platform Configuration Methods*.
2. Locate the following parameter and set it to `true`:

   ```
   "feature.databricks.connection.enabled": true,
   ```
3. To allow for direct publishing of job results to Databricks tables from the Run Job page, you must enable the following parameters. For more information on these settings, see *Run Job Page*.

Parameter	Description
feature.databricks. enableDeltaTableWrites	Set this value to `true` to enable users to choose to write generated results to Databricks delta tables from the Run Job page.
feature.databricks. enableExternalTableWrites	Set this value to `true` to enable users to choose to write generated results to Databricks external tables from the Run Job page.

4. Save your changes and restart the platform.

Create Connection

This connection can also be created via API. For details on values to use when creating via API, see *Connection Types*.

Please create an Databricks connection and then specify the following properties with the listed values:

> **NOTE:** Host and port number connection information is taken from Azure Databricks and does not need to be re-entered here. See *Configure for Azure Databricks*.

Property	Description
Connect String options	Please insert any connection string options that you need. Connect String options are not required for this connection.

Test Connection	Click this button to test the specified connection.
Default Column Data Type Inference	Set to `disabled` to prevent the Trifacta platform from applying its own type inference to each column on import. The default value is `enabled`.

Use

For more information, see *Using Databricks Tables*.

Data Conversion

For more information on how values are converted during input and output with this database, see *Databricks Tables Data Type Conversions*.

Troubleshooting

Failure when importing wide Databricks Tables table

If you are attempting to import a table containing a large number of columns (>200), you may encounter an error message similar to the following:

```
2019-11-26T10:17:11.439Z [XNIO-3 task-15] ERROR com.trifacta.dataservice.DataServiceController -
[sid=de7f0e78-087d-4922-9251-337c0cc6da71] - [rid=4851532f-3fb1-4746-9884-a1f71c8209ee] - [method=POST] -
[url=/jdbc/datastream] - Stack trace: com.trifacta.dataservice.connect.exception.UnknownJdbcException: A
system error occurred: org.apache.spark.SparkException: Job aborted due to stage failure: Task 0 in stage
408.0 failed 4 times, most recent failure: Lost task 0.3 in stage 408.0 (TID 1342, 10.139.64.11, executor
11): org.apache.spark.SparkException: Kryo serialization failed: Buffer overflow. Available: 0, required:
1426050. To avoid this, increase spark.kryoserializer.buffer.max value.
```

The problem is that the serializer ran out of memory.

Solution:

To address this issue, you can increase the Kyroserializer buffer size.

1. You can apply this change through the *Admin Settings Page* (recommended) or `trifacta-conf.json`. For more information, see *Platform Configuration Methods*.
2. Locate the `spark.props` section and add the following setting. Modify `2000` (2GB) depending on whether your import is successful:

   ```
   "spark.kryoserializer.buffer.max.mb": "2000"
   ```

3. Save your changes and restart the platform.
4. Attempt to import the dataset again. If it fails, you can try incrementally raising the above value.

For more information on passing property values into Spark, see *Configure for Spark*.