databricks

Best practices for building an integration with Databricks

Authors : Databricks Technology Partner Team
Last updated : May 8, 2024

Introduction

This document is a best practices guide for ISV and Technology partner products to integrate with Databricks.

It includes guidance on choosing appropriate architecture, APIs & compute for integration and using the Databricks APIs in accordance with best practices.

Designing a Databricks integration involves the following high level steps:

- Create a Databricks account and a workspace on AWS, Azure or GCP
  - Create your Databricks account. Link
  - Create a unity catalog enabled Databricks Workspace. Link to docs on AWS, Azure, GCP
- Based on product category or integration use cases
  - Identify appropriate Databricks API / SDK/ connector to use for building the integration
  - Use this document to learn best practices for integrating with Databricks

- Ensure user-agent and telemetry metadata is passed on all Databricks API calls

Guided workflows for Technology Partner's building integrations with Databricks

Introduction

Technology partners and Independent software vendors (ISVs) build integrations to connect their products and services to Databricks Lakehouse platform. This can be done through a variety of methods such as jobs, api's, connectors, sdk's and others.

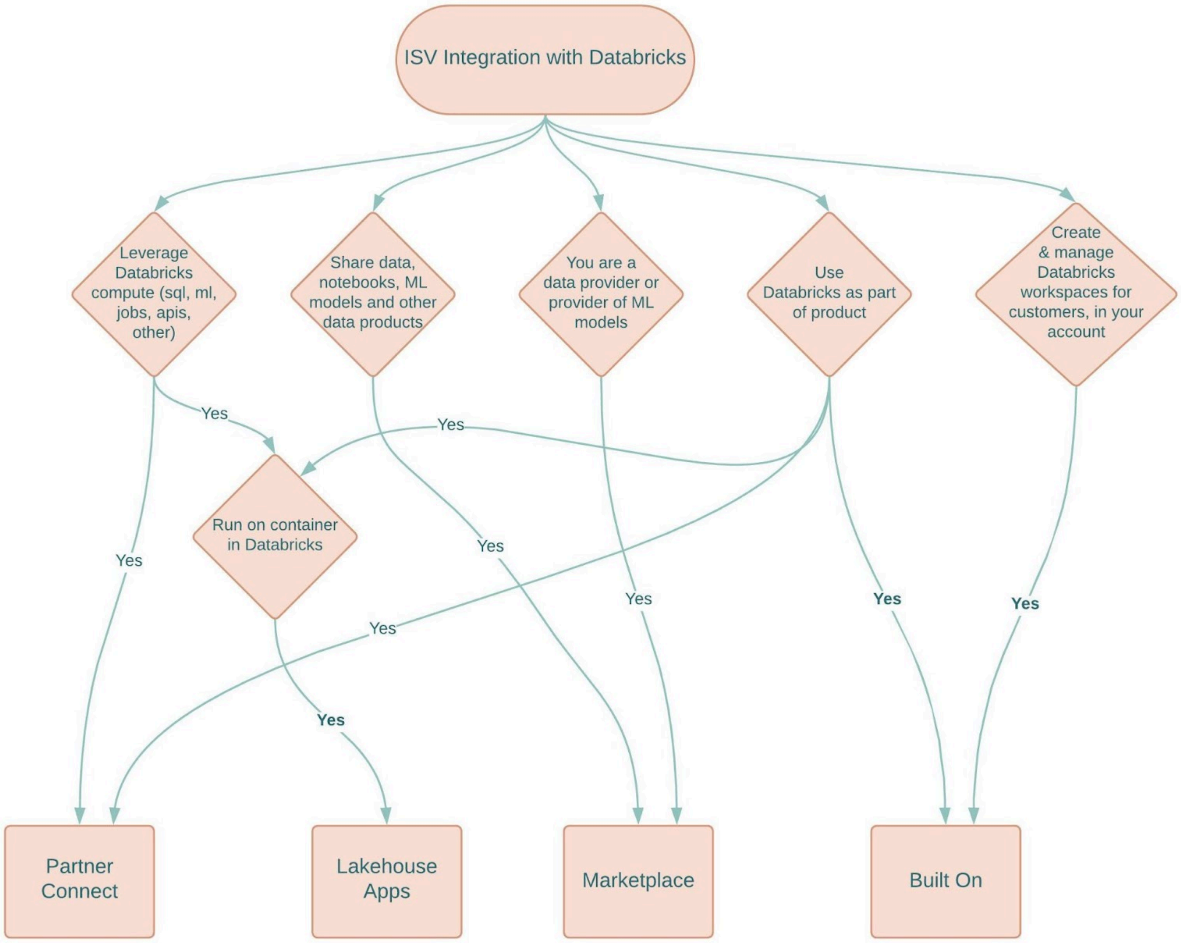
What next : Available programs

Once an integration has been built and validated by the Technology Partner team, you are eligible to participate in one or more of the following programs.

- Technology partnership
  - Partner Connect
- Built On
- Data partnership
  - Marketplace
- Lakehouse Apps

Based on the technology partner's product's functionality and integration with Databricks, below illustrates at a high level the programs the partner product is eligible to.

Let's take an example of an ISV product that focuses on ELT / Ingest / Transform. They would likely use one of Databricks connectors or sdk's to push their jobs as SQL and use a Databricks SQL warehouse for compute. Depending on how the product is architected, they could be eligible to participate in partner connect, lakehouse apps and built-on.


ISV Integration with Databricks
Leverage Databricks compute (sql, ml, jobs, apis, other)
Share data, notebooks, ML models and other data products
You are a data provider or provider of ML models
Use Databricks as part of product
Create & manage Databricks workspaces for customers, in your account
Run on container in Databricks
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Partner Connect
Lakehouse Apps
Marketplace
Built On

1. Partner Connect

Partner Connect allows customers/users to connect their Databricks workspace to partner solutions from the Databricks UI. This gives customers/users the ability to try out partner solutions using their data in the Databricks Lakehouse, and then adopt the solutions that best meet your business needs.

Partner Connect provides a simpler alternative to manual partner connections, by provisioning the required Databricks resources on their behalf and passing resource details to the partner. Not all Databricks partner solutions are featured in Partner Connect.

- Partner Connect | Databricks
- API requirements: GitHub - databrickslabs/partner-connect-api

ISV partner products that typically fall under this program include : ingest, transform, ETL, reverse ETL, BI, visualization, monitoring, semantic layer, governance, security, data science & ml/ai, and others.

Pre-requisites

a. Integrate with Databricks Unity Catalog
b. Have a SaaS based trial experience for the ISV product
c. Leverage Databricks compute by executing jobs, push down sql/python/scala workloads, ml/mlflow integrations, call Model Serving endpoints, calling Databricks apis, etc.
d. This is invite only. Work with your Databricks Partner Development Manager on the eligibility, requirements and next steps.
    i. Have “X” joint customers using the integration.

2. Marketplace

Databricks Marketplace is an open marketplace for data products, including datasets, notebooks, machine learning models, and more. It allows data providers to share data products securely, and data consumers to explore and expand their access to the data and data services they need.

Data products in the Marketplace are powered by the open-source Delta Sharing standard, which ensures that data is shared securely and reliably.

Data products in Databricks Marketplace can be either public or private. Public data products are available to anyone with a Databricks account, while private data products are only available to members of a specific private exchange.

- Databricks Marketplace
- Introducing Databricks Marketplace, an Open Marketplace for Data Solutions - The Databricks Blog
- Documentation: What is Databricks Marketplace?
- https://marketplace.databricks.com/

ISV partner products who typically fall under this program: share data, notebooks, ML models or data products.

Prerequisites

a. Use Delta Sharing
b. Integrate with Databricks Unity Catalog
c. Share data, notebooks, ML models or data products

3. Lakehouse Apps

Lakehouse Apps are a new way to build, distribute, and run innovative data and AI applications directly on the Databricks Lakehouse Platform. Lakehouse Apps are built with the technology of your choice. Apps run on secure, auto-scale compute that runs containerized code that can be written in virtually any language, so developers are not limited to building in any specific framework.

Lakehouse Apps are fully integrated with the Databricks Lakehouse Platform, so they can access all of your data, leverage all of the Databricks services, and be managed and governed using the same tools and processes as your other Databricks workloads.

Lakehouse Apps can be distributed through the Databricks Marketplace, so they can be easily discovered and adopted by other Databricks users.

- Introducing Lakehouse Apps | Databricks Blog

Prerequisites

a. Any ISV product that has integration with Databricks
b. Invite only
c. Coming soon
d. Run on containers

4. Built On

The Databricks Built On program is a great way for independent software vendors (ISVs) to build and deliver reliable, scalable, and secure data and AI solutions. ISVs can build their applications or products on top of the Databricks Lakehouse Platform, which means that the ISV product is leveraging Databricks as part of its core product.

Reference for key terminologies

Databricks Account

- A Databricks account is a top-level entity that represents a user or organization.
- A Databricks account represents a single entity that can include multiple workspaces. Accounts enabled for Unity Catalog can be used to manage users and their access to data centrally across all of the workspaces in the account. Billing and support are also handled at the account level.
- Getting started guide for Databricks account and workspace

Workspace

- In Databricks, a workspace is a Databricks deployment in the cloud that functions as an environment for your team to access Databricks assets. Your organization can choose to have either multiple workspaces or just one, depending on its needs.

User

- Part of the Databricks identity management model. User identities recognized by Databricks and represented by email addresses.

Service principal

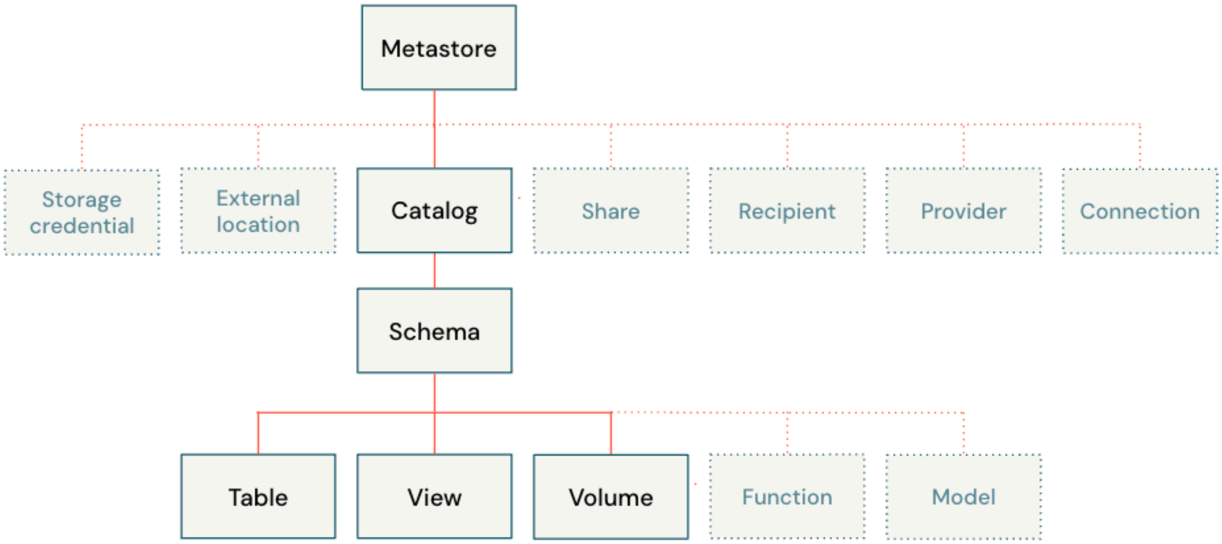
- Part of the Databricks identity management model. Identities for use with jobs, automated tools, and systems such as scripts, apps, and CI/CD platforms.

Unity Catalog

- Unity Catalog provides centralized access control, auditing, lineage, and data discovery capabilities across Databricks workspaces.
- Getting started guide for Databricks Unity Catalog. [RECOMMENDED] Use this guide to setup your workspace powered by Databricks Unity Catalog

Unity Catalog object model

- In Unity Catalog, the hierarchy of primary data objects flows from metastore to table or volume
- The hierarchy of primary data objects in Unity Catalog. See graph below: Link to doc around Unity Catalog object model.



Catalog

- ○ Part of the Unity Catalog object model. The first layer of the object hierarchy, used to organize your data assets(for example Schemas, Tables, and Volumes)

Schema

- ○ Part of the Unity Catalog object model. Also known as databases, schemas are the second layer of the object hierarchy and contain tables and views.

Table

- ○ Part of the Unity Catalog object model. A table resides in the third layer of Unity Catalog's three-level namespace. It contains rows of data.

Integrating with Databricks: Available api's , connectors and sdk's

Databricks provides numerous options that once can use to build product integrations with Databricks. One can use a specific mechanism depending on the architecture of your product, the use case you are trying to address and the programming language that you use in your product/application.

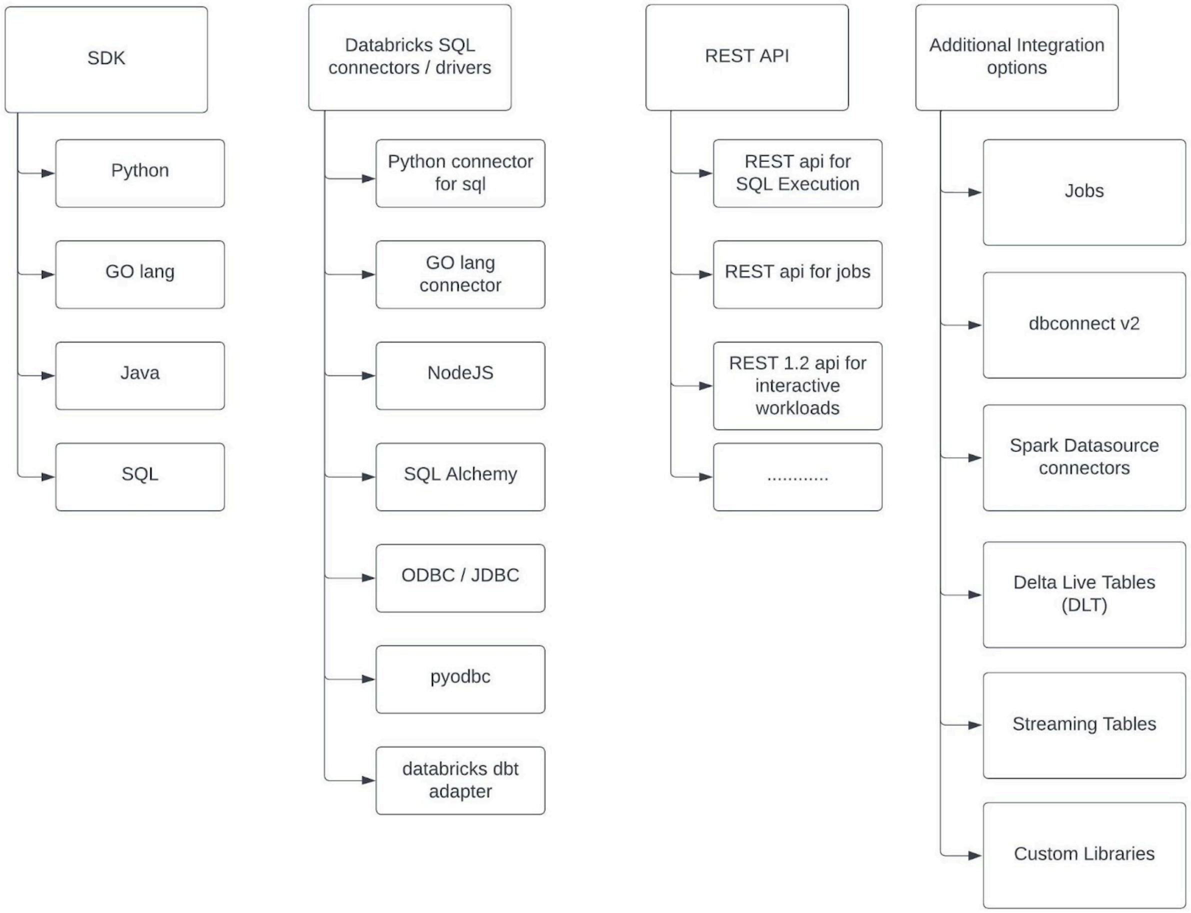
For example, a BI product will use a Databricks JDBC/ODBC or Databricks SQL connectors to push down SQL queries to a Databricks SQL warehouse. A visual ETL product, which generates python or Scala code, will use Databricks REST APIs to submit Databricks Jobs that execute on jobs compute.

Below are some examples of products and how they integrate

ISV Product Type	Example ISV	Use Case
BI	Tableau / PowerBI	Use ODBC and Push down SQL
Visual ETL that generates SQL	Matillion	Use JDBC and Push down SQL
Visual ETL that generates Scala/Java	Prophecy	Use Rest 1.2 API and Push down pyspark/scala jobs
Visual ETL - Interactive/IDE	VS code, Dataiku,Matlab	Use Databricks connect
Enterprise Catalog	Collibra, Alation	Use rest api and JDBC+SQL to retrieve Databricks metadata
Data Integration/Connectors	Fivetran, Rivery, Hevodata, Arcion	Use JDBC and SQL to Ingest data into Databricks
ETL products that generate/execute dbt projects	Dbt cloud, Fivetran, Prophecy	Use Databricks dbt adapter to pushdown SQL

Note that some ISVs implement multiple use cases within the same product - for example, a visual ETL product might retrieve table metadata via JDBC to populate a list of tables in their UI AND also push down scala/python jobs via REST APIs.

Integrating with Databricks : Available options



Databricks has extensive support for SQL, Python, Rest api's. The product documentation provides details and examples for all the above. This document aims to provide some guidance on that to use for some common integration scenarios. If you have architecture questions or specific needs, reach out to your Databricks Partner Manager or Partner Solutions Architect (request to schedule an architecture review session).

Lets deep dive into some of the key mechanisms to integrate with Databricks

SQL connectors for Databricks

You can use SQL connectors, drivers, and APIs to connect to and run SQL statements and commands from Databricks compute resources. These SQL connectors, drivers, and APIs include:

- SQL connector for python
- SQL Driver for GO
- SQL Driver for NodeJs
- SQL statement execution api (REST)
- Pyodbc
- ODBC driver
- JDBC driver

Learn more

- https://docs.databricks.com/dev-tools/index-driver.html
-

Databricks Connect (aka DbConnect)

Databricks Connect is a client library for the Databricks Runtime. It allows you to write jobs using Spark APIs and run them remotely on a Databricks cluster instead of in the local Spark session. Examples of when to use this include Interactive IDE's, notebooks, custom applications, interactive execution of jobs.

Learn more

- https://docs.databricks.com/dev-tools/databricks-connect.html

Databricks Jobs

A Databricks job is a way to run your data processing and analysis applications in a Databricks workspace. Your job can consist of a single task or can be a large, multi-task workflow with complex dependencies. Databricks manages the task orchestration, cluster management, monitoring, and error reporting for all of your jobs. You can run your jobs immediately, periodically through an easy-to-use scheduling system.

You can package your job to be executed as a jar or a python whl file and then create/invoke the job using the Rest api.

The following are two ways to create the jobs

- Use the “create & run now” api to create jobs. These jobs are visible on the Databricks ui and there is a limit on the total number of jobs on a workspace (default is 1000 jobs)
- Use the “Runs Submit” api to create jobs via api. These jobs will not be visible on the databricks ui and you will have to use the rest api to query for the status of the jobs. Jobs submitted by this api do not count towards the maximum 1000 jobs per workspace.
    -

Learn more

- https://docs.databricks.com/workflows/index.html#what-is-databricks-jobs
- https://docs.databricks.com/workflows/jobs/jobs-2.0-api.html
-

Examples of integration mechanism to use

Let's look at a couple of examples on how you should think about building integrations with Databricks. More prescriptive guidance around various use cases are described in later sections of this document.

Example 1:

If your application architecture uses SQL for ingest and pushdown for transformations, recommend using the SQL apis executed via JDBC/ODBC/other connectors

- Connect to a Databricks SQL warehouse or interactive cluster via ODBC/JDBC/connectors/SDKs and then execute SQL. For example execute SQL for create table, copy into, insert , merge and others
- Use SQL to ingest data into Databricks
- Use SQL to push down transformations

Example 2:

If your application architecture prefers integrating via REST apis and submitting jobs (python,scala,java, jars) to Databricks, then use REST 2.0/1.2 apis.

- REST 2.0 api is used for example to submit jobs using python, scala or java.
- REST 1.2 api is the preferred api to handle interactive execution. For example similar to what you can do on the cells in a databricks notebook.

More on these below

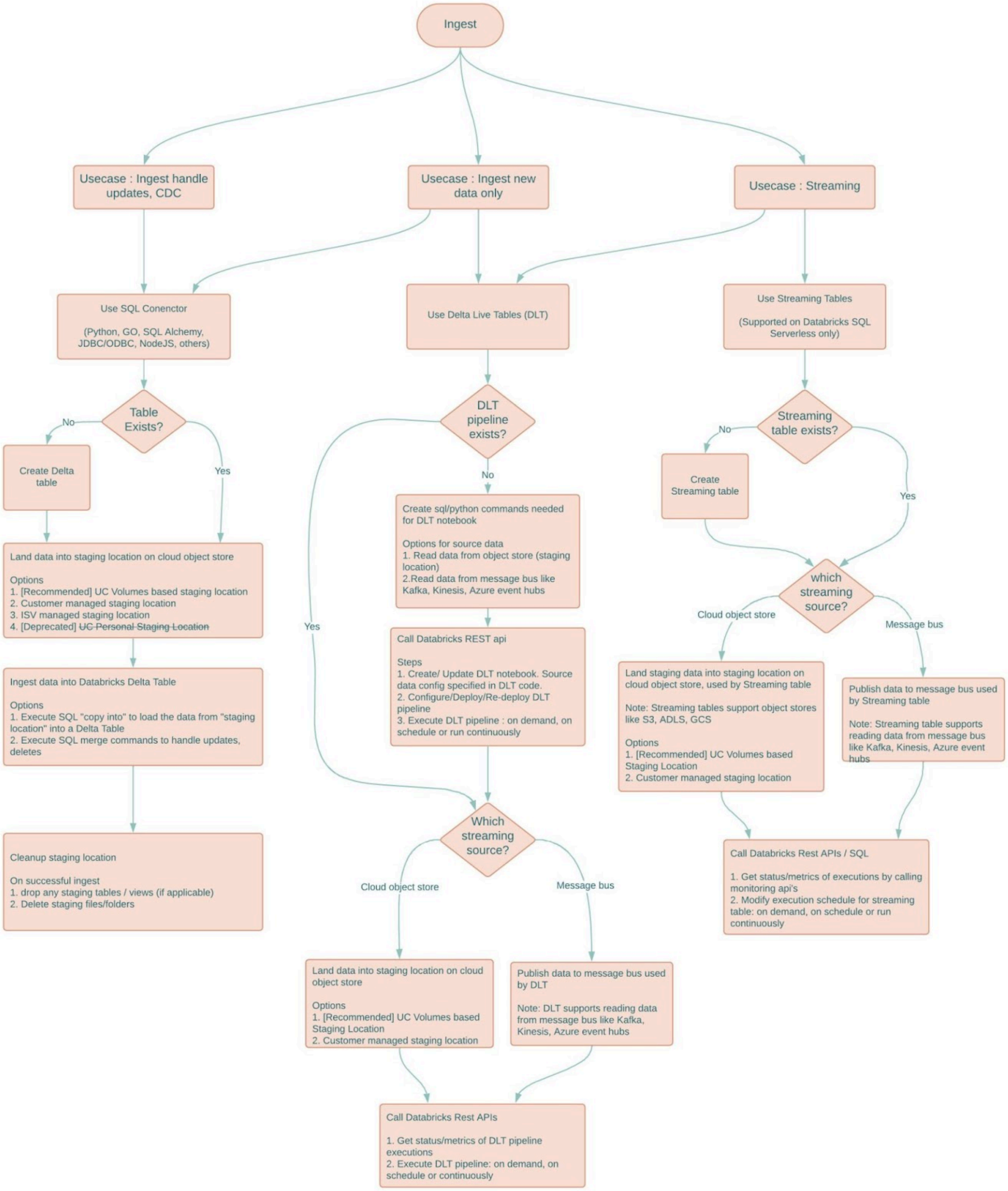
- Rest API 2.0
    - Supports submission of batch and structured streaming jobs, coded in either python or scala/java (packaged as jar files).
    - Submitted jobs can run on an all purpose cluster or a job cluster
    - Not a good fit for SQL push down
    - Not a good fit for applications that require interactive SLAs
    - For python & scala jobs this is the recommended API, *unless* you must support an interactive SLA (see Rest 1.2 API section below)

- Rest API 1.2 (command execution api)
    - Supports submission of python and scala snippets to an all purpose cluster (job clusters not supported)
    - Good fit for interactive SLAs (for eg, if a human is waiting on a UI for a result on the client side)
    - Not a good fit for batch or structured streaming jobs
- Databricks connect
    - DB conenct would be another option to consider if you are using python/scala/r in an interactive manner
    - It currently does not support pushing down multi node or single node ML jobs
    -

Recommendations based on product categories

Ingest, CDC, Streaming Ingest, Data Replication

Below describes the high-level workflow associated with building an integration with Databricks for an ISV product to support the following use cases : Ingest, Change Data Capture (CDC), Streaming Ingest and data replication.



Usecase : Ingest handle updates, CDC

Use SQL Conenctor

(Python, GO, SQL Alchemy, JDBC/ODBC, NodeJS, others)

Create Delta table

Land data into staging location on cloud object store

Options
1. [Recommended] UC Volumes based staging location
2. Customer managed staging location
3. ISV managed staging location
4. [Deprecated] ~~UC Personal Staging Location~~

Ingest data into Databricks Delta Table

Options
1. Execute SQL "copy into" to load the data from "staging location" into a Delta Table
2. Execute SQL merge commands to handle updates, deletes

Cleanup staging location

On successful ingest
1. drop any staging tables / views (if applicable)
2. Delete staging files/folders

Usecase : Ingest new data only

Use Delta Live Tables (DLT)

Create sql/python commands needed for DLT notebook

Options for source data
1. Read data from object store (staging location)
2.Read data from message bus like Kafka, Kinesis, Azure event hubs

Call Databricks REST api

Steps
1. Create/ Update DLT notebook. Source data config specified in DLT code.
2. Configure/Deploy/Re-deploy DLT pipeline
3. Execute DLT pipeline : on demand, on schedule or run continuously

Land data into staging location on cloud object store

Options
1. [Recommended] UC Volumes based Staging Location
2. Customer managed staging location

Publish data to message bus used by DLT

Note: DLT supports reading data from message bus like Kafka, Kinesis, Azure event hubs

Call Databricks Rest APIs

1. Get status/metrics of DLT pipeline executions
2. Execute DLT pipeline: on demand, on schedule or continuously

Usecase : Streaming

Use Streaming Tables

(Supported on Databricks SQL Serverless only)

Create Streaming table

Land staging data into staging location on cloud object store, used by Streaming table

Note: Streaming tables support object stores like S3, ADLS, GCS

Options
1. [Recommended] UC Volumes based Staging Location
2. Customer managed staging location

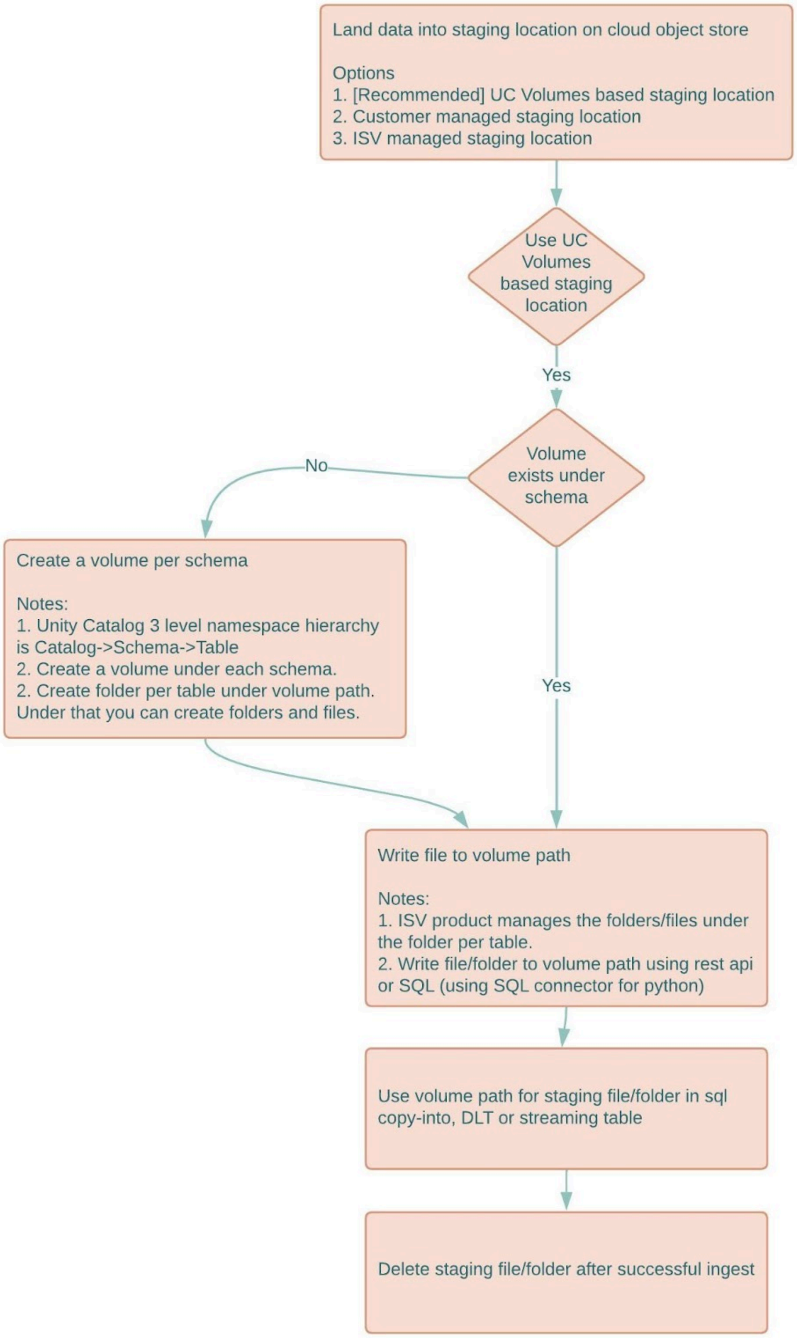
Publish data to message bus used by Streaming table

Note: Streaming table supports reading data from message bus like Kafka, Kinesis, Azure event hubs

Call Databricks Rest APIs / SQL

1. Get status/metrics of executions by calling monitoring api's
2. Modify execution schedule for streaming table: on demand, on schedule or run continuously

Below describes the high-level workflow on how to use Unity Catalog (UC) Volumes as a temporary staging location for data.


Land data into staging location on cloud object store
Options
1. [Recommended] UC Volumes based staging location
2. Customer managed staging location
3. ISV managed staging location
Use UC Volumes based staging location
Yes
Volume exists under schema
No
Create a volume per schema
Notes:
1. Unity Catalog 3 level namespace hierarchy is Catalog->Schema->Table
2. Create a volume under each schema.
2. Create folder per table under volume path. Under that you can create folders and files.
Yes
Write file to volume path
Notes:
1. ISV product manages the folders/files under the folder per table.
2. Write file/folder to volume path using rest api or SQL (using SQL connector for python)
Use volume path for staging file/folder in sql copy-into, DLT or streaming table
Delete staging file/folder after successful ingest

Transformation, ETL, Data prep

Visual low code data prep

Monitoring & Observability

BI and Visualization

Security

Data Science, ML and AI

Refer to the section on Data Science, ML and AI integrations

REST API best practices

Some rest api best practices when iterating with Databricks

- Always set the “user-agent” HTTP header on REST API and JDBC/ODBC calls . This needs to your “[isv_vendor_name]_[product name]”
- If submitting Jobs via REST API, use runs-submit API
- Do not use DBFS API to move large amounts of data or as a staging location for Delta ingest for production workloads
- Use Databricks volumes for larger amounts of data or as a staging location for Delta ingest for production workloads
- Build retries into all REST API calls to handle 500, 429 & 503 response codes

    - Upto 10 minutes, at 30s intervals
- Build Job retry options on your UIs
    - Sometimes a job fails due to temporary issues - customers like to retry automatically

Relevant REST API & Workspace upper limits

Component	Limit	Comments
Jobs API input payload size	10 MB	
Jobs API - Create (Visible on UI) per workspace	1000	Use Runs Submit API to bypass this limitation
Concurrent running jobs per workspace	1000	
Number of jobs run per hour in a workspace	5000	
Job Runs list expiration	60 days	Export runs if needed beyond 60 days
Metastore connection limit per workspace	Guaranteed 100 today, 200 default by Q3	If limits hit, use external metastore
REST API Requests per second per workspace	30	Build in retries to handle the 429 error code when limits are reached
Rest 1.2 API result payload limit	2MB	
Rest 1.2 API input payload limit	10 kb	

Designing the UI for the Databricks connector

When designing the connection screen on your product to connect to Databricks, the following is some high level guidance.

Connection details

These can be obtained from the Databricks Compute UI or Databricks SQL Warehouses UI. Passing the UserAgent attribute is required for all ISV integrations with Databricks. The UserAgent tag is passed as part of the connection request. (additional details are available in the section around UserAgent). Here is an example of an ISV designed Databricks connection dialog.

Providing an ability to pass additional JDBC/ODBC attributes would be useful for some customers. Example of additional advanced attributes include proxy server configs, logging, timeouts, UseNativeQuery and others advanced options specified in the Databricks Simba Driver documentation

- Hostname : The hostname for the Databricks workspace
- Port : Default is 433
- HTTP Path or JDBC URI

Note: One can set "UseNativeQuery=1" to ensure the driver does not transform the queries. Not needed if using the latest databricks drivers

Authentication

These are the authentication mechanisms supported by Databricks. We support OAuth currently on Azure using Azure Active Directory and OAuth on AWS and GCP. Refer to the OAuth integration guide for additional details.

- OAuth [Recommended]
    - Azure Active Directory
        - On Azure we support OAuth using Azure AAD
        - This is the preferred mode of authentication for Azure Databricks
        - Additional fields required : AAD endpoint
    - AWS
    - GCP

- Personal Access Token (PAT Token) [Required]
    - This is the preferred mode for authentication on AWS and GCP
    - PAT tokens are supported on Azure Databricks

- Username / Password
    - The following two options are available if using this option to authenticate
    - Use the username of the Databricks user and their password.
    - Use username = "token" and password =" value of PAT token"

Advanced Configurations

Ingest and ELT integrations would require additional configurations like location for staging data and location of the target table. Customers might start with managed Delta tables but would move to using unmanaged Delta tables (data stored directly on cloud storage location in s3/adls/gcs).

- Catalog name
  Databricks unity catalog name could be entered by a user, this catalog name can then be set on the connection level as the default. This ensures that you can support one catalog set at the connection level. If your product already supports 3 level namespaces then you may or may not specify this field.
- Staging location
  The staging location for data to be ingested into Delta can either use a location that is managed by the ISV product or managed by the customer.

  - Databricks Volumes based staging location
    - This is managed by Databricks and secured using the unity catalog
    - Refer to the section on how to use Volumes for staging data fro ingest
  - Managed by ISV
    - Staging folder is managed by the ISV, in their own account.
    - ISV product to use AWS STS tokens during ingestion into Delta, using the “copy into” command.
  - Customer managed
    - Customer provides an s3/adls/gcs location where the ISV product can land the staging data.
    - Customer provides access credentials to ISV product to write staging data.
    - Databricks cluster or SQL endpoint is configured to read from the staging location.
- Table data location
  Databricks customers tend to use cloud storage location for Delta tables (i.e external tables or unmanaged tables). They could provide a base path that could then used when creating tables
  - Databricks compute or SQL warehouse is configured to read/write to the table data location.
  - Refer to the section on UC Storage location, Credential, and assigning the ACL’s to the user

○

Integration checklist - Ingesting data into Databricks

ISV ingest product integrations should support the following features when creating tables

- Support 3 level namespace : catalog_name, schema_name and table_name
- Auto create delta tables when ingesting data
- Add appropriate table metadata. Refer to section around “Table creation”
- Support managed delta tables
- [optional] Support unmanaged delta tables
- [optional] Set delta optimization on tables
- Pass UserAgent tag for all calls to Databricks REST /JDBC / ODBC /Connector/ SDK calls. Refer to the section around setting UserAgent.
- Implement the merge optimization best practices. Refer to section around “Merge best practices”

Table creation and Unity Catalog metadata

Table and Column properties

Setting table comments and column comments allows setting useful metadata about a table. If a comment was already set, it overrides the old value with the new one.

For example passing a comment at the time of creating the table

```
CREATE or REPLACE TABLE democatalog.mydatabase.events (
  date DATE COMMENT 'The date the event took place',
  eventId STRING COMMENT 'The Id for the event',
  eventType STRING COMMENT 'The event type',
  data STRING COMMENT 'Data about the event')
USING DELTA
PARTITIONED BY (date)
LOCATION '/tmp/delta/events'
COMMENT 'A table comment.'
```

To set a comment on an existing table

```
ALTER TABLE democatalog.mydatabase.events SET TBLPROPERTIES ('comment' = 'A table
comment.')
```

To set a comment on the column for an existing tables

```
ALTER TABLE democatalog.mydatabase.events ALTER eventType COMMENT 'The event type.
updated'
```

Requirements around order of columns as you create table

Make sure that the primary key, foreign keys and other key columns are within the first 32 columns. Delta collects statistics only for the first 32 columns. These statistics help improve performance of queries and merges.

Setting primary and foreign key metadata

Typical ISV products that ingest data into Databricks, first extract data from source systems like databases, applications, streaming systems, files and others. They introspect the schemas from the source systems and create the target Databricks delta tables. It is recommended that primary keys and foreign key relationships are applied to the Databricks tables. Databricks stores the metadata on the primary key and foreign keys and currently does not enforce the constraints.

- If there are primary key/ foreign keys, make sure to add the metadata for the tables on Databricks
  - https://docs.databricks.com/sql/language-manual/sql-ref-syntax-ddl-create-table-constraint.html
  - https://docs.databricks.com/sql/language-manual/sql-ref-syntax-ddl-alter-table-add-constraint.html

Prerequisite : Unity Catalog should be enabled and used on the workspace.

Generated columns in Delta

Delta supports generated columns which are a special type of columns whose values are automatically generated based on a user-specified function over other columns in the Delta table. When you write to a table with generated columns and you do not explicitly provide values for them, Delta Lake automatically computes the values. For example, you can automatically generate a date column (for partitioning the table by date) from the timestamp column; any writes into the table need only specify the data for the timestamp column.

Additional links to relevant documentation

- https://docs.databricks.com/delta/generated-columns.html

Example:

```
CREATE TABLE democatalog.democatalog.mydatabase.events (
eventId bigint,
eventTime timestamp,
eventDate date GENERATED ALWAYS AS ( CAST(eventTime AS DATE) ) )
USING DELTA
PARTITIONED BY (eventDate)
```

Handling identity columns and surrogate keys

Additional links to relevant documentation

- https://www.databricks.com/blog/2022/08/08/identity-columns-to-generate-surrogate-keys-are-now-available-in-a-lakehouse-near-you.html

Enabling Deletion Vectors

- Enable Deletion Vectors to speed up MERGE performance.
- Requires MERGE being run using photon-enabled or warehouse clusters.
    - Works with DBR 12.1+ with Photon
    - DBSQL pro and DBSQL serverless
    - Call cluster rest api to get the above details
- [RECOMMENDED] Enable Deletion Vectors by setting the table property:

```Java
ALTER TABLE <table_name>  SET TBLPROPERTIES
('delta.enableDeletionVectors' = true);
```

Prerequisite : Unity Catalog should be enabled and used on the workspace. Photon needs to be enabled on the DBSQL pro or DBSQL serverless compute.

Resources: Table creation and data types

- [How to Identity Columns to Generate Surrogate Keys in the Databricks Lakehouse]()
- [What's a Dimensional Model and How to Implement It on the Databricks Lakehouse Platform]()

Managed vs external tables

Refer to [External tables | Databricks on AWS]() for details around external tables

Managed Delta tables

For a managed table both the data and the metadata are managed. Doing a DROP TABLE deletes both the metadata and data.

Example:

```
CREATE IF NOT EXISTS TABLE democatalog.mydatabase.events (
  date DATE,
  eventId STRING,
  eventType STRING,
  data STRING)
```

```
USING DELTA
```

External tables

For external tables you specify the LOCATION as a path. Tables created with a specified LOCATION are considered external or unmanaged by the metastore. Unlike a managed table, where no path is specified. An external table's files are not deleted when you DROP the table.

Example:

```
CREATE TABLE democatalog.mydatabase.events (
  date DATE,
  eventId STRING,
  eventType STRING,
  data STRING)
USING DELTA
LOCATION '/[s3/adls/mnt]/delta/events'
```

If you want to overwrite the data at a location for Delta tables then you can use the following

```
CREATE OR REPLACE TABLE democatalog.mydatabase.events (
  date DATE,
  eventId STRING,
  eventType STRING,
  data STRING)
USING DELTA
LOCATION 's3a://delta/events'
```

If a table with the same name already exists, the table is replaced with the new configuration. Databricks strongly recommends using CREATE OR REPLACE instead of dropping and re-creating tables.

Important Notes

1. "CREATE OR REPLACE" is supported for Delta
2. If you have created an external table, drop the table and try to recreate the unmanaged table using the same location, the second create table will fail. You will have to make sure that the location is empty before running second create table
3. If you have created an external table, and you want to overwrite data+schema, then you can use the " CREATE OR REPLACE TABLE". This overwrites the table+schema+data at the specified location.

The location specified can be a s3 or adls or gcp paths. For Example

Amazon S3 path

```
s3a://bucket/path/to/dir
```

Azure adls gen2 path

```
abfss://<file-system-name>@<storage-account-name>.dfs.core.windows.net/<directory-name>/path/to/dir
```

UC Volume path

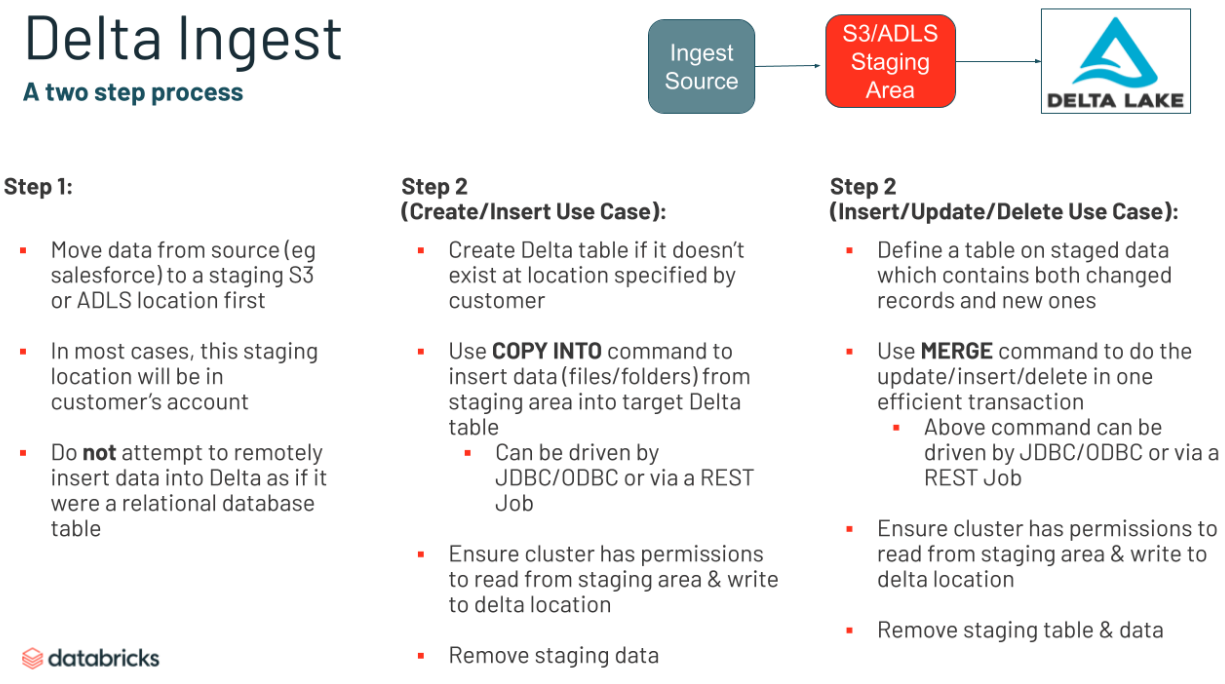
UC Personal staging location path

Ingest newly arriving data with no updates

Some of the options for integration include the following

- Using SQL “copy into”
- Using Delta Live Tables (DLT)
- Using Streaming tables

At a high level below provides the steps to ingest data into Delta



Delta Ingest

A two step process

Ingest Source → S3/ADLS Staging Area → DELTA LAKE

Step 1:

- Move data from source (eg salesforce) to a staging S3 or ADLS location first
- In most cases, this staging location will be in customer's account
- Do **not** attempt to remotely insert data into Delta as if it were a relational database table

Step 2 (Create/Insert Use Case):

- Create Delta table if it doesn't exist at location specified by customer
- Use **COPY INTO** command to insert data (files/folders) from staging area into target Delta table
  - Can be driven by JDBC/ODBC or via a REST Job
- Ensure cluster has permissions to read from staging area & write to delta location
- Remove staging data

Step 2 (Insert/Update/Delete Use Case):

- Define a table on staged data which contains both changed records and new ones
- Use **MERGE** command to do the update/insert/delete in one efficient transaction
  - Above command can be driven by JDBC/ODBC or via a REST Job
- Ensure cluster has permissions to read from staging area & write to delta location
- Remove staging table & data

databricks

Using SQL “copy into”

Using the “Copy Into” command is the preferred method to ingest newly arriving data into Databricks. It does not handle updates to data in the table.

The ”Copy into” requires the data to be staged as files, on cloud storage or uc volumes. Use any format supported by spark as the file format for the staged data. If you have the ability to write the staging data as Parquet, then Parquet would be the preferred file format.

There are four options for staging data

- UC Volumes based staging location [Recommended]
- UC Personal staging location based staging location [Deprecated]
- ISV managed staging S3 bucket
  - The ISV uses their own S3 bucket to stage data for ingestion (for their customers).
  - Use AWS STS tokens and SSE-E encryption when invoking the “copy into” command. Details on using these options in the next section
- Customer provided staging location
  - Customer to provide credentials to be able to write to the staging folder
  - Databricks cluster configured to be able to read data from the staging location

Example of using the copy command using a csv staging file

```
COPY INTO demo_catalog.demo_db.demo_users
FROM (select id, first_name, last_name, email, gender, ip_address FROM 's3://pkona-isv-staging/')
FILEFORMAT = CSV
PATTERN = 'mock_data/demo_users_csv/mock_data_v1_correct_col_order.csv'
FORMAT_OPTIONS('header' = 'true' , 'inferSchema' = 'true')
COPY_OPTIONS ('force' = 'false')
```

Example of using the copy command using a parquet staging file

```
COPY INTO demo_catalog.demo_db.atm_transactions
FROM 's3://pkona-isv-staging/mock_data/atm_transactions_parquet'
FILEFORMAT = PARQUET
PATTERN = 'part-00001*.snappy.parquet'
COPY_OPTIONS ('force' = 'false')
```

Additional examples on how to use the copy command available in sample notebooks provided as part of this document bundle.

Additional links to relevant documentation

- https://docs.databricks.com/spark/latest/spark-sql/language-manual/delta-copy-into.html

Using Delta Live Tables (DLT)

Refer to the section under Delta Live Tables (DLT)

Using Streaming tables

Refer to the section under streaming tables

Merge : Best practices for performance optimization

This section describes some to the performance optimization best practices when doing merge:

1. Use a DBSQL Pro or Serverless Warehouse or a Photon-enabled Cluster whenever possible. Use the most recent DBR version when using a Cluster.

2. Z-Order the target table by the join keys of the merge. This has two benefits:

   a. It reduces the number of files that a merge statement has to rewrite. For many merges only a small subset of the “key space” is affected, and by z-ordering the table we can restrict this small subset to a small number of files.
   b. It also allows additional optimizations to be added that allow pruning the target table.

3. Prune the files in the target table that need to be read by the merge by adding additional filters to the join condition of the merge.

   As an example, consider the following merge statement that performs an upsert:

   ```
   MERGE INTO target t
   USING source s
   ON t.pk1 = s.pk1 AND … AND t.pkn = s.pkn
   WHEN MATCHED THEN UPDATE SET *
   WHEN NOT MATCHED THEN INSERT *
   ```

   First we need to collect the filter values that we are going to add:

   ```
   SELECT MIN(pk1), MAX(pk1), …, MIN(pkn), MAX(pkn)
   FROM source
   ```

   Then, using the result of this query, we need to modify the merge statement as follows:

   ```
   MERGE INTO target t
   USING source s
   ON t.pk1 = s.pk1 AND … AND t.pkn = s.pkn
   AND t.pk1 >= {min_pk1} AND t.pk1 <= {max_pk1}
   AND …
   AND t.pkn >= {min_pkn} AND t.pkn <= {max_pkn}
   WHEN MATCHED THEN UPDATE SET *
   WHEN NOT MATCHED THEN INSERT *
   ```

4. Ensure that the join keys are part of the first 32 columns of both the target and the source table. Delta currently only collects min/max statistics for the first 32 columns in the table. Without these statistics we cannot prune the target table, and we cannot efficiently collect the min and max of the columns in the source table.
5. If you plan to run multiple concurrent merges on the same table, make sure to have no conflicts. Check the links below for details
   a. https://docs.databricks.com/optimizations/isolation-level.html

b. https://docs.databricks.com/optimizations/isolation-level.html#avoid-conflicts-using-partitioning-and-disjoint-command-conditions
c. https://docs.databricks.com/optimizations/isolation-level.html#write-conflicts-on-databricks

Ingesting data using Delta Live Table (DLT)

What is Delta Live Tables is fully explained in our documentation starting with defining the differences between a streaming table, Materialized view and views. Start with this page to understand the fundamentals.

Limitations

The following limitations apply:

- All tables created and updated by Delta Live Tables are Delta tables.
- Delta Live Tables tables can only be defined once, meaning they can only be the target of a single operation in all Delta Live Tables pipelines.
- Identity columns are not supported with tables that are the target of APPLY CHANGES INTO and might be recomputed during updates for materialized views. For this reason, Databricks recommends only using identity columns with streaming tables in Delta Live Tables. See Use identity columns in Delta Lake.

Resources

- Delta Live Tables has full support in the Databricks REST API. See Delta Live Tables API guide.
- For pipeline and table settings, see Delta Live Tables properties reference.
- Delta Live Tables SQL language reference.
- Delta Live Tables Python language reference.

System Limits

Resource	Limit
Notebooks per DLT Pipeline	25 limit, can be raised on request
Concurrent DLT Pipelines per Workspace	100 limit, can be raised on request

Scalability Guidelines

Resource	Guidance on Table Count
# of live tables in a triggered pipeline	< 100
# of streaming live tables in a triggered pipeline	< 50
# of live + streaming tables in a continuous pipeline	< 25

A DLT pipeline can easily be managed following the REST API Pipeline reference once the notebook has been imported. Alternatively a notebook can be imported using the Workspace command of the Databricks CLI or the Databricks Terraform provider prior to scheduling your DLT pipeline.

DLT workshop git repo https://github.com/ddubeau/DLT-hands-on-workshop

Examples of a DLT pipeline with Auto Loader

1. Load a notebook via Api using Basic Authentication (see Authentication for Databricks automation for additional information)

```
curl —netrc –X POST \
htttps://<databricks-instance>/api/2.0/workspace-files/import \
 --header "Authorization: Bearer $databricks-token" \
 --header ‘Content-type: multipart/form-data’ \
 --form path=/Repos/ddubeau/DLT-hands-on-workshop \
 --form format=SOURCE \
 --form language=SQL \
 --form content=02. Building Delta Live Tables Pipelines-SQL
 --form overwrite=true
```

Replace:

- <databricks-instance> with the Databricks workspace instance name, for example dbc-a1b2345c-d6e7.cloud.databricks.com
- $databricks-token with the Databricks token

2. Create a DLT Pipeline

```
curl --netrc -X POST \
https://<databricks-instance>/api/2.0/pipelines \
--data @pipeline-settings.json
```

3. Start a DLT pipeline

```
curl --netrc -X POST \
https://<databricks-instance>/api/2.0/pipelines/<pipeline-id>/updates \
--data '{ "full_refresh": "true" }'
```

For a full list of API examples, please refer to the Delta Live Table API guide.

SQL “Copy Into” best practices

Using AWS STS and SSE-C encryption with “Copy Into”

STS authentication

This feature is only supported with our new S3A client, and if multiple credential sets will be used for the same bucket filesystem caching must be disabled. To configure this, add the following Spark configuration settings to the cluster:

```
spark.hadoop.fs.s3.impl shaded.databricks.org.apache.hadoop.fs.s3a.S3AFileSystem
spark.hadoop.fs.s3a.impl shaded.databricks.org.apache.hadoop.fs.s3a.S3AFileSystem
spark.hadoop.fs.s3n.impl shaded.databricks.org.apache.hadoop.fs.s3a.S3AFileSystem
spark.hadoop.fs.s3.impl.disable.cache true
spark.hadoop.fs.s3a.impl.disable.cache true
spark.hadoop.fs.s3n.impl.disable.cache true
```

STS authentication allows accessing an S3 bucket with temporary credentials generated through the AWS Security Token Service. Long-lived credentials are unsupported, as they should not be embedded in SQL queries. Usage:

```
COPY INTO delta.`/my/target/path` FROM '/my/source/path'
    FILEFORMAT = CSV
    CREDENTIALS (
        'awsKeyId' = '$key',
        'awsSecretKey' = '$secret',
        'awsSessionToken' = '$token
    )
```

SSE-C encryption

This feature requires the same Spark configuration as STS authentication.

SSE-C encryption allows storing files in encrypted form to AWS and securely reading them into the cluster running COPY INTO. This can be helpful for preventing cross-reads in shared source buckets. To use this feature, store the source files into AWS using SSE-C, and then run:

```
COPY INTO delta.`/my/target/path` FROM '/my/source/path'
    FILEFORMAT = CSV
    ENCRYPTION ('type' = 'SSE-C', 'masterKey' = '$encryptionKey')
    CREDENTIALS (
        'awsKeyId' = '$key',
        'awsSecretKey' = '$secret',
        'awsSessionToken' = '$token
    )
```

User Agent Tag Prerequisite for all Integrations

An user agent tag is to be passed by all partner integrations with Databricks. This tag supports tracking of usage which helps improve customer satisfaction and increase customer adoption of the integration.

Passing the user agent tag by integrations that use REST API, ODBC or JDBC is required for any partner integration

- To be certified by Databricks
- To be part of the Databricks Partner Gallery
- To be part of the Databricks Databricks Data Ingest Network

Passing HTTP User Agent Tag for REST API's

Here is the detail on the HTTP user agent that needs to be populated on the rest calls to databricks api's

- HTTP User-Agent is present in all request headers when calling Databricks REST apis.The user-agent should have the name of the name of the isv (and or integration name). https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent
  - For example: HTTP User-Agent should have <isv-name+product-name> in the user agent string text

Passing User Agent Tag for ODBC/JDBC

User application name and version

Simba, starting with JDBC v2.6.16 and ODBC v2.6.15 drivers , has added a new connection string parameter, through which user applications can specify their entry to the User-Agent. The parameter will be **UserAgentEntry**. Simba will validate the value provided by the user application against the format <isv-name+product-name>/<product-version> <comment>
Only one comment is allowed, without any nesting - ie. any characters except for comma, parentheses or new lines.
Spaces in connection string values are preserved without any escaping.

Simba will return an error to the user application if the entry does not conform: "Incorrect format for User-Agent entry, received: <value obtained from connection string>"

For example

- Tableau could specify: `Tableau/2020.1 (Databricks)` for Databricks datasource connector or `Tableau/2020.1 (Spark SQL)` for Spark SQL datasource connector

Example 1 : Sample code from Tableau connector

Example connection-builder.js Code snipped used by Tableau connector that passes the User Agent Tag

```
/*
Databricks Tableau Connector
Copyright 2019 Databricks, Inc.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/

(function dsbuilder(attr) {
  var params = {};

         // The Databricks cluster ODBC endpoint
         params["HOST"] = attr["server"];
         params["PORT"] = "443";
         params["HTTPPATH"] = attr["dbname"];
         params["THRIFTTRANSPORT"] = "2";
         params["SPARKSERVERTYPE"] = "3";
         params["SSL"] = "1";

         // Authentication by username and password only
         params["AUTHMECH"] = 3;
         params["UID"] = attr["username"];
         params["PWD"] = attr["password"];

         // Use the native HiveQL query emitted by Tableau
         params["USENATIVEQUERY"] = "1";

         // Automatically reconnect to the cluster if an error occurs
         params["AutoReconnect"] = "1";

         // Minimum interval between consecutive polls for query execution status (1ms)
         params["AsyncExecPollInterval"] = "1";

         // Tell the ODBC driver that it is Tableau connecting.
         params["UserAgentEntry"] = "Tableau";

         var formattedParams = [];
  formattedParams.push(
                 connectionHelper.formatKeyValuePair(
                              driverLocator.keywordDriver, driverLocator.locateDriver(attr)));

    for (var key in params) {
      formattedParams.push(connectionHelper.formatKeyValuePair(key, params[key]));
    }

    return formattedParams;
  })
```

Example 2 : To set the user agent for JDBC in Java

Sample code for setting the User Agent

```
com.simba.spark.jdbc.DataSource ds = new com.simba.spark.jdbc.DataSource();
ds.setCustomProperty("UserAgentEntry", "<isv-name+product-name>");
```

Example 3: To set the user agent for JDBC as part of the JDBC URI

Append ";UserAgentEntry=<isv-name+product-name>" to the connection URL (the one that starts with "jdbc:spark://").

Passing User Agent as a cluster config

☐ This approach uses spark configs to pass the user agent tag. This approach can be used only after being approved by the Databricks ISV team.

On a cluster set the following spark config property

- `spark.databricks.isv.product=<list_of_comma_seperated_values_of_isv_partner_names>`
    - `for example if two isv names are used on a cluster then "spark.databricks.isv.product=isv_name1,isv_name2"`

Passing User Agent Tag when using Databricks connectors

Here is how you can pass the correct ISV user-agent. ISV's can set a string as user-agent (<isv-name+product-name>)

Passing user agent tag for Databricks SQL Connectors

Check the example in the section “Canceling queries and passing user agent” on how to pass user agent for the Databricks connectors

- Go connector
- Node.js connector
- Python connector
- JDBC
- ODBC

Passing user agent tag for Databricks Connect

The “user_agent” should be added as part of the connection, as part of the connection parameters. Link . In addition, if you are using the DatabricksSession from dbconnect you can use DatabricksSession.builder.userAgent("my_useragent")....getOrCreate()

Using Databricks Connectors

Canceling queries and passing user agent

Here is how you can pass the correct ISV user-agent; cancel operation. ISV’s can set a string as user-agent<isv-name+product-name>.

Go connector

<<Content coming soon>>

Node.js connector

```
async function initClient({ host, endpointId, token, client }) {
    const client = new DBSQLClient();
    return client.connect({
        host,
        path: `/sql/1.0/endpoints/${endpointId}`,
        token,
        clientId: `<isv-name+product-name>`,
    });
}
…
    const client = await initClient(host, endpoint, token, client);
    const session = await client.openSession();
    const queryOperation = await session.executeStatement(
      `
        SELECT id
        FROM RANGE(100000000)
        ORDER BY RANDOM() + 2 asc
      `,
      { runAsync: true },
    );
    queryOperation.cancel();
```

Python connector

```
        databricks.sql.connect(**self.DUMMY_CONNECTION_ARGS,
_user_agent_entry=”<isv-name+product-name>")

          with self.cursor({}) as cursor:
            def execute_really_long_query():
                cursor.execute("SELECT SUM(A.id - B.id) " +
                               "FROM range(1000000000) A CROSS JOIN range(100000000) B " +
                               "GROUP BY (A.id - B.id)")
            exec_thread = threading.Thread(target=execute_really_long_query)
            exec_thread.start()
            cursor.cancel()
```

JDBC

```
url="jdbc:databricks://adb-111111111111xxxxx.xx.azuredatabricks.net:443/default;transportMode=http;ssl=1;httpPath=sql/protocolv1/o/<workspaceId>/<clusterId>;AuthMech=11;Auth_Flow=0;Auth_AccessToken={0};UserAgentEntry=<isv-name+product-name>".format(access_token)

val connection = DriverManager.getConnection(url, user, "")
val statement = connection.createStatement()
val f = Future {
  statement.executeQuery("SELECT COUNT(*) FROM big_table")
}
```

```
Thread.sleep(1000)

statement.cancel()
```

ODBC

```
    SQLHDBC dbc;
    char conn[1024] = "";
    char *buffer = stpcpy(stpcpy(conn, "DRIVER="), PATH);
    buffer = stpcpy(stpcpy(buffer, ";UserAgentEntry="), "<isv-name+product-name>");
    buffer = stpcpy(stpcpy(buffer, ";HOST="), SHARD);
    buffer = stpcpy(stpcpy(buffer, ";PORT="), "443");
    buffer = stpcpy(stpcpy(buffer, ";AuthMech="), "3");
    buffer = stpcpy(stpcpy(buffer, ";HTTPPath="), HTTP_PATH);
    buffer = stpcpy(stpcpy(buffer, ";UID="), "token");
    buffer = stpcpy(stpcpy(buffer, ";PWD="), PWD);
    buffer = stpcpy(stpcpy(buffer, ";SSL="), "1");
    buffer = stpcpy(stpcpy(buffer, ";ThriftTransport="), "2");
    //buffer = stpcpy(stpcpy(buffer, ";RowsFetchedPerBlock="), rowsPerFetch); // 10000
    buffer = stpcpy(stpcpy(buffer, ";EnableArrow="), withArrow); // hidden
    buffer = stpcpy(stpcpy(buffer, ";EnableQueryResultDownload="), "1"); //hidden 1
    buffer = stpcpy(stpcpy(buffer, ";UseNativeQuery="), "1"); //hidden 1
    buffer = stpcpy(stpcpy(buffer, ";EnableCurlDebugLogging="), "1");
    buffer = stpcpy(stpcpy(buffer, ";LogLevel="), "4");
    printf("Connecting string: %s\n", conn);
    printf("************************************\n");
    SQLDriverConnect(dbc, NULL, conn, SQL_NTS, NULL, 0, NULL, SQL_DRIVER_COMPLETE);
...
    SQLHSTMT stmt;
...
    SQLCancel(stmt);
```

Data science, ML & AI Integrations

Integrating with Foundational Model API's / LLM's on Databricks

What are foundation models?

Foundation models are large ML models pre-trained with the intention that they are to be fine-tuned for more specific language understanding and generation tasks. These models are utilized to discern patterns within the input data for generative AI and LLM workloads.

Databricks Model Serving supports serving and querying foundation models using the following capabilities:

- Foundation Model APIs. This functionality makes state-of-the-art open models available to your model serving endpoint. These models are curated foundation model architectures that support optimized inference. Base models, like Llama-2-70B-chat, BGE-Large, and Mistral-7B are available for immediate use with pay-per-token pricing, and workloads that require performance guarantees and fine-tuned model variants can be deployed with provisioned throughput.

- External models. These are models that are hosted outside of Databricks. Endpoints that serve external models can be centrally governed and customers can establish rate limits and access control for them. Examples include foundation models like, OpenAI's GPT-4, Anthropic's Claude, and others.

What are Databricks Foundation Model APIs?

Databricks Model Serving now supports Foundation Model APIs which allow you to access and query state-of-the-art open models from a serving endpoint. With Foundation Model APIs, you can quickly and easily build applications that leverage a high-quality generative AI model without maintaining your own model deployment.

The Foundation Model APIs are provided in two pricing modes:

- Pay-per-token: This is the easiest way to start accessing Foundation Models on Databricks and is recommended for beginning your journey with Foundation Models.

- Provisioned throughput: This mode is recommended for workloads that require performance guarantees, fine-tuned models, or have additional security requirements.

Query foundational models

To call foundational models on Databricks, you can utilize any one of the following

- OpenAI client
- REST API
- MLflow Deployments SDK
- Databricks GenAI SDK
- SQL function
- Langchain

Refer to doc link for more details and examples

- https://docs.databricks.com/en/machine-learning/model-serving/score-foundation-models.html

Query provisioned throughput endpoint

Refer to doc link for more details and the direct link to an example

- https://docs.databricks.com/en/machine-learning/foundation-models/deploy-prov-throughput-foundation-model-apis.html#notebook-examples

Below is a code snippet in python (from above notebook link) to call a provisioned throughput endpoint

```
data = {
   "inputs": {
       "prompt": [
           "Below is an instruction that describes a task. Write a response that appropriately completes the request.\n\n### Instruction:\nWhat is Apache Spark?\n\n### Response:\n"
       ]
   },
   "params": {
       "max_tokens": 100,
       "temperature": 0.0
   }
}

headers = {
   "Context-Type": "text/json",
   "Authorization": f"Bearer {API_TOKEN}"
}

response = requests.post(
   url=f"{API_ROOT}/serving-endpoints/{endpoint_name}/invocations",
   json=data,
   headers=headers
)

print(json.dumps(response.json()))
```

Query models on databricks using SQL

SQL ai_query() function

The ai_query() function is a built-in Databricks SQL function, part of AI functions. It allows these types of models to be accessible from SQL queries:

- Custom models hosted by a model serving endpoint.
- Models/LLM's hosted by Databricks Foundation Model APIs.
- External models (third-party models hosted outside of Databricks).

Example: Query a large language model

The following example queries the model behind the sentiment-analysis endpoint with the text dataset and specifies the return type of the request.

```
SELECT text, ai_query(
    "sentiment-analysis",
    text,
    returnType => "STRUCT<label:STRING, score:DOUBLE>"
  ) AS predict
FROM
  Catalog.schema.customer_reviews
```

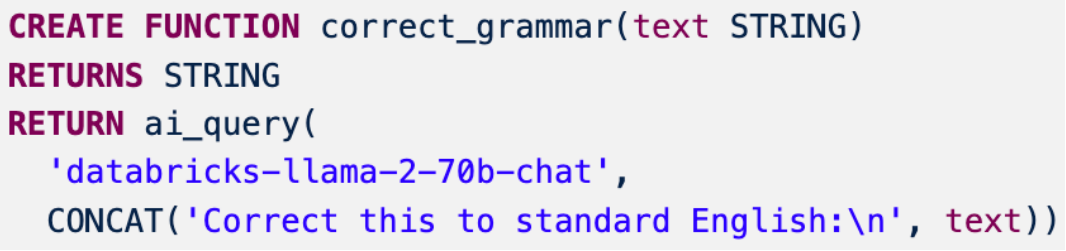
Example: Query a predictive model

The following example queries a classification model behind the spam-classification endpoint to batch predict whether the text is spam in “inbox_messages” table. The model takes 3 input features: timestamp, sender, text. The model returns a boolean array.

```
SELECT text, ai_query(
  endpoint => "spam-classification",
  request => named_struct(
    "timestamp", timestamp,
    "sender", from_number,
    "text", text),
  returnType => "BOOLEAN") AS is_spam
FROM catalog.schema.inbox_messages
```

Example: Creating your own SQL function that uses ai_query()

The following example creates your own custom SQL function that leverages ai_query() to call the llama-2-70b foundational model. This example function takes a string as input , leverages the LLM returns a corrected english string.



```
CREATE FUNCTION correct_grammar(text STRING)
RETURNS STRING
RETURN ai_query(
  'databricks-llama-2-70b-chat',
  CONCAT('Correct this to standard English:\n', text))
```

For additional details checkout

- https://docs.databricks.com/en/large-language-models/how-to-ai-query.html
- https://docs.databricks.com/en/sql/language-manual/functions/ai_query.html

SQL AI functions

Databricks AI Functions are built-in SQL functions that allow you to apply AI on your data directly from SQL. These functions invoke a state-of-the-art generative AI model from Databricks Foundation Model APIs to perform tasks like sentiment analysis, classification and translation.

For additional details checkout

- https://docs.databricks.com/en/large-language-models/ai-functions.html
- https://docs.databricks.com/en/large-language-models/ai-functions-example.html

REST API : Get details on serving endpoints

Get all serving endpoints

- https://docs.databricks.com/api/workspace/servingendpoints/list
- If you are interested in Foundational Model api's or LLM's, then check for the
  - "endpoint_type": "FOUNDATION_MODEL_API" or
  - "type": "FOUNDATION_MODEL",

Get a single serving endpoint

- https://docs.databricks.com/api/workspace/servingendpoints/get

Python SDK :Get details on serving endpoints

Example code on getting list of foundational models and external models

- https://github.com/prasadkona/databricks-integration-examples/blob/main/src/model_serving/using-databricks-python-sdk-model-serving-endpoints.py

Integrating with Databricks Vector Search

What is Databricks Vector Search?

Databricks Vector Search is a vector database that is built into the Databricks Intelligence Platform and integrated with its governance and productivity tools. A vector database is a database that is optimized to store and retrieve embeddings.

With Vector Search, you create a vector search index from a Delta table. The index includes embedded data with metadata. You can then query the index using a REST API or Python SDK or LangChain, to identify the most similar vectors and return the associated documents. You can structure the index to automatically sync when the underlying Delta table is updated.

For more on this refer to https://docs.databricks.com/en/generative-ai/vector-search.html

Query a Vector Search endpoint

Examples on how to call a Databricks Vector Search endpoint using below are available at https://docs.databricks.com/en/generative-ai/create-query-vector-search.html#query-a-vector-search-endpoint

- Python SDK (link to sample notebook)
- REST api
- LangChain (link to LangChain docs)

Models in Unity Catalog

Getting started with models in unity catalog

Models in Unity Catalog extends the benefits of Unity Catalog to ML models, including centralized access control, auditing, lineage, and model discovery across workspaces. Models in Unity Catalog is compatible with the open-source MLflow Python client.

Getting started is simple

- Reference models using their three-level naming
    - *<catalog>.<schema>.<model>*

- Models in Unity Catalog are compatible with the MLflow Python client.

- To upgrade ML workflows to target Unity Catalog, simply: Configure MLflow client to target Unity Catalog

  import mlflow
  mlflow.set_registry_uri("databricks-uc")

For additional details checkout
https://docs.databricks.com/en/machine-learning/manage-model-lifecycle/index.html

Registering a MLflow Model to Databricks

You can register a MLflow model to Databricks Managed MLflow Model Registry using the `mlflow` python package. If you are registering from a MLflow tracking service that you manage, configure MLflow as you would normally. Then set environment variables for access to Databricks and set the registry URI to `databricks`. After that you can call `register_model` (docs), passing in the model URI for the run's model.

```
import os
import mlflow

os.environ['DATABRICKS_HOST'] = <DATABRICKS WORKSPACE URL>
os.environ['DATABRICKS_TOKEN'] = <DATABRICKS TOKEN>
```

```
mlflow.set_registry_uri("databricks-uc")
mlflow.register_model('runs://0/d97600326268486daa7e187eee8fc3a2/model',
<MODEL NAME>) # model nem in the format <catalog>.<schema>.<model>
```

If you instead have the file on local disk you can use a model URI using the file schema:

```
mlflow.register_model('file:path-to-model-directory', <MODEL NAME>)
```

For additional details checkout
https://docs.databricks.com/en/machine-learning/manage-model-lifecycle/index.html

Downloading a MLflow Model from Databricks

To download a MLflow model from Databricks you can use the MLflow python client.
You can set the Databricks authentication and host through

```
import os
os.environ['DATABRICKS_HOST'] = <DATABRICKS WORKSPACE URL>
os.environ['DATABRICKS_TOKEN'] = <DATABRICKS TOKEN>
```

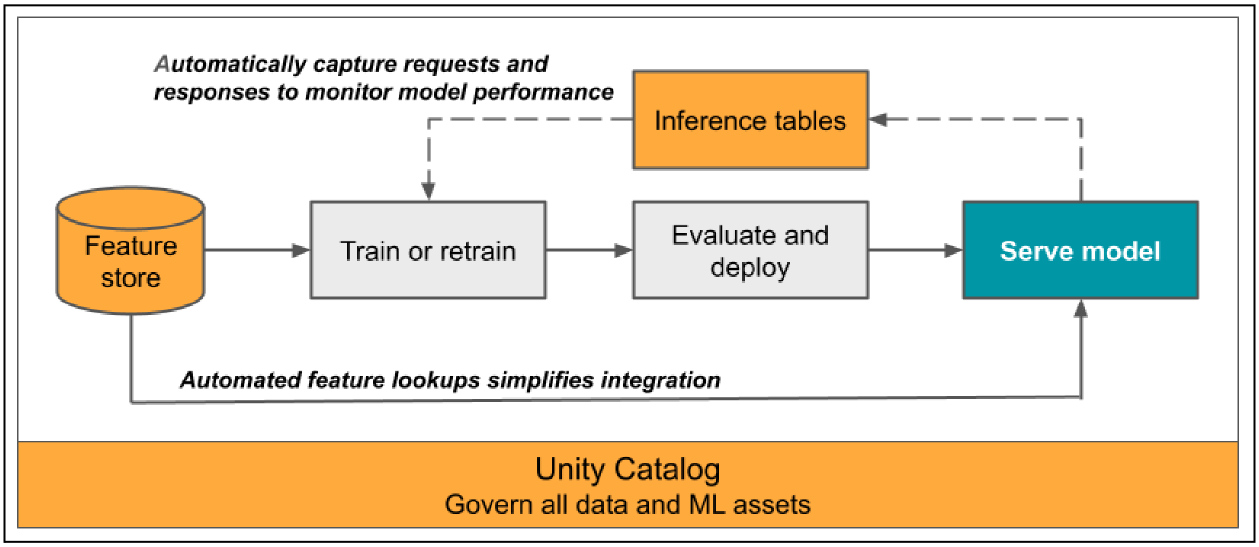
Then set the tracking or registry URI to `databricks` using
`mlflow.set_tracking_uri('databricks')` or
`mlflow.set_registry_uri('databricks-uc')` depending on whether you are downloading the model from the tracking service or the model registry.

Finally, call mlflow.artifacts.download_artifacts to download the model.

Monitoring LLM’s and Models on Databricks

Inference Tables

Inference table automatically captures incoming requests and outgoing responses for a model serving endpoint and logs them as a Unity Catalog Delta table. You can use the data in this table to monitor, debug, and improve ML models.


Automatically capture requests and responses to monitor model performance
Inference tables
Feature store
Train or retrain
Evaluate and deploy
Serve model
Automated feature lookups simplifies integration
Unity Catalog
Govern all data and ML assets

To learn more about Inference tables

- https://docs.databricks.com/en/machine-learning/model-serving/inference-tables.html

Handling Images

For large image files (average image size greater than 100 MB), Databricks recommends using the Delta table only to manage the metadata (list of file names) and loading the images from the object store using their paths when needed.

It is recommended to store the image files (both large and small) on Unity Catalog Volumes. The delta table stores the path to the image file i.e UC volumes path, s3/adls/gcs path or http path.

For images and files, Databricks allows you to use the binary file data source to load image data into the Spark DataFrame as raw bytes. See Reference solution for image applications for the recommended workflow to handle image data.

In summary, the two ways customers store images on Databricks

1. Delta table which contains file metadata including path where actual file is stored (UC volumes path, s3/adls/gcs path or http path)

2. Delta table having a column of type binary to store the content of the image file (Check the reference solution notebook above). Not recommended for files larger than 100MB)

Guidance based on product category

Data Labeling and Generation (Human & Synthetic)

Basic

- Publish data from Databricks to labeling tool
    - Recommended approach:
        - For text data, we recommend Delta tables or UC volumes or Delta Sharing.
        - For image or other binary data, we recommend a combination of a metadata table along with references to paths in distributed file storage (UC Volumes - recommended, s3/adls/gcs path, http path).
            - Check section on “Handling Images”
        - Partner tools can use databricks connectors, sdk’s or drivers to query for data on databricks.
        - Delta Sharing should be considered if the customer wants to securely share a single copy of data with a partner tool. No need to push data to partner tool
    - Alternative approach:
        - Use an upload API to publish data to your product, and provide an example notebook for doing so. For example, provide a python library.
- Publish data from the partner tool into Databricks
    - Recommended approach:
        - Directly publish tables or updates to tables to the customer’s account as labeling happens. See the “Ingesting data into Delta” section of ISV Integration Best Practices and “Unity Catalog Guide for ISV Partners”
            - Allow an admin to specify a schema within a catalog to publish labels generated
            - Ingest the data into a table on databricks
                - Leverage databricks sdk or connector or driver
                - Upload staging file to uc volumes (using the put api’s provided by the connector/sdl/driver)
                - Execute SQL Copy-into the target table
    - Alternative approach:
        - Provide a library the customers can use on Databricks to pull your APIs and create a Spark Dataframe they can write to Delta.
- Provide example notebooks and documentation

    - Use standard label formats wherever possible - e.g formats compatible with popular model training tools.

Advanced

- Integrate into item selection, model assisted labeling, human-in-the-loop workflows, through sample notebooks
- Integrate as part of the RAG studio workflow
- Integrate with Partner Connect

Natural Language Processing

Basic

- Model training and/or fine tuning can be run directly in Databricks or invoked from Databricks
- Model training and/or fine tuning metrics/parameters logged to MLflow tracking service, preferably through autologging. Consider working with the MLflow open source community to add autologging capabilities for your libraries, especially for open source libraries.
- Models can be logged as a MLflow model, either through the built-in flavors specific to your library (or add support for your library), or the base python function support.
- Logged models can be used in batch inference on Spark.
- Provide sample notebooks for these operations

Advanced

- Logged models can be used with Serverless Real Time Inference(SRTI)
    - Avoid requiring Java library, as SRTI currently does not support Java library dependencies.

Model Training Tools or Model hubs

Basic

- Examples and documentation for loading and using models for inference on Databricks
- If applicable, examples and documentation for fine tuning models on Databricks
- Models can be logged as a MLflow model, either through the built-in flavors specific to your library (or add support for your library), or the base python function support.

Advanced

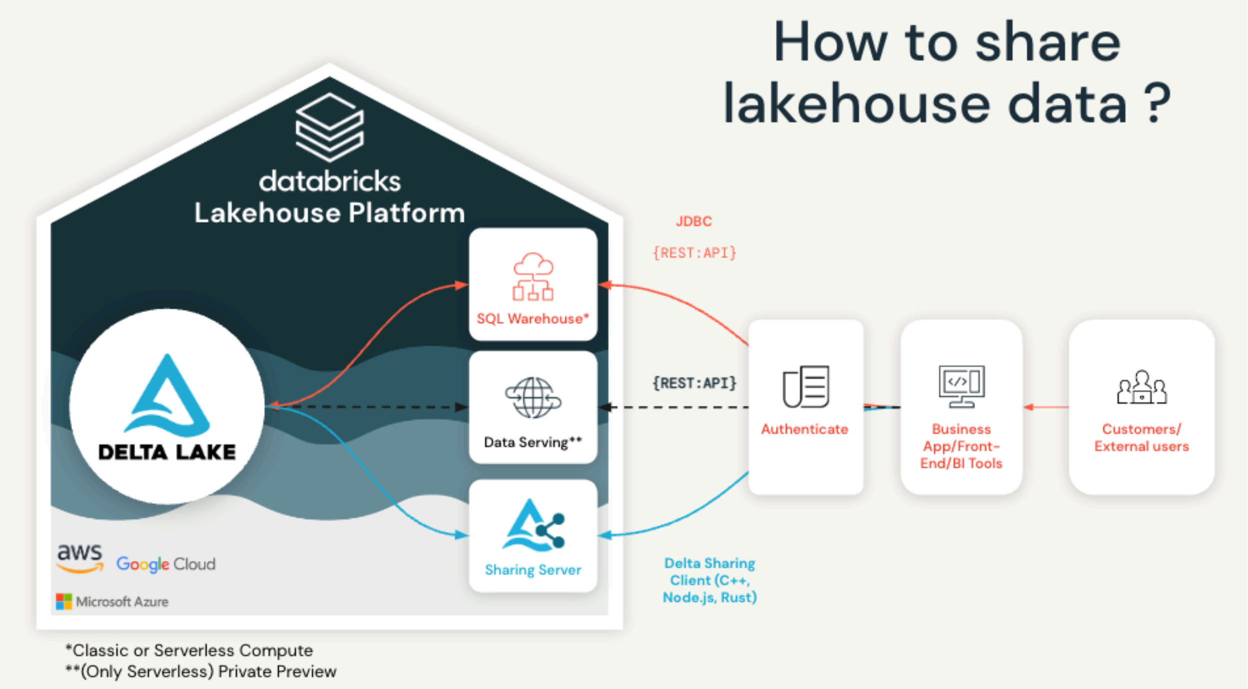
- Models can be registered to Databricks' Model Registry directly from your tool
- Publish the models to Databricks marketplace (for models)

Reference Blogs and links

Misc
https://www.databricks.com/blog/accelerating-your-deep-learning-pytorch-lightning-databricks

Big book of ml-ops
https://www.databricks.com/resources/ebook/the-big-book-of-mlops

Data Sharing - Delta Sharing



How to share lakehouse data ?

*Classic or Serverless Compute
**(Only Serverless) Private Preview

SQL	Delta Sharing	REST API
Data formats supported: • Delta, Iceberg, Hudi, Parquet, csv, and more **Costs:** • Provider pays compute cost • Provide pays egress cost (small)	**Data formats supported:** • Delta only **Costs:** • Consumer pays the compute cost • Provider pays egress cost **Additional**	**Data formats supported:** • Delta, Iceberg, Hudi, Parquet, csv, and more **Costs:** • Provider pays compute cost • Provide pays egress cost (small)

Additional considerations: - Data set size is limited - Data will be ingested sequentially - Only supports SQL **Authentication:** - Databricks platform token	**considerations:** - No Data size limit - Replication recommended for AZ distance limitations **Authentication:** - Within the Databricks platform, secured by the metastore IDs and databricks VPN - Outside of the platform Delta Sharing credential token	**Additional considerations:** - Data set size is limited - Data will be ingested sequentially - Leverages ML Serving as REST API endpoint - Accessible anywhere import package can be run - No dependencies on Databricks Partner team to launch **Authentication:** - Databricks platform token

Sharing via SQL

There are different ways to access the data in the Lakehouse via SQL. These are described in SQL based integrations section.

Sharing via REST API

With the new ML Serving and the upcoming data serving service you can serve your data via REST API directly from your Lakehouse.
ML Serving relies on Databricks Serverless offering, there's a need to enable it first in the account console.

See the Model Inference public docs for more information

Sharing via Delta Sharing

As a Provider

Delta Sharing support sharing tables and will soon support other data assets such as notebooks, models, files, and more
The Databricks platform offers a managed Delta Sharing service at no additional cost.

Requirement

- You have to have Unity Catalog configured and a metastore attached to the workspace
- You enabled sharing on your metastore
- As of now Delta format is the only format supported by Delta Sharing. It is easy to autoload your data into Delta
- Your storage cannot prevent external access. You should not allow any public access, but have to enable access via IAM role, Firewalls, or other settings.

Sharing can be done from your Databricks platform to your consumer Databrick platform. The instructions are detailed here
Sharing can be done outside of your consumer Databrick plastform as well. You can restrict the access to a predefined network CIDR range if neede. Detailed instructions can be found here

Optional configuration:

You can create an isolated environment to share the data.

- A separate bucket to hold the data being shared. There's no need to create another copy of the table, your core table can resides in this bucket - ready to be shared

As a Consumer

The data shared with you can be consumed in different ways.
The easiest way is to get the data connected in your Databricks metastore. You'll need a Unity Catalog metastore to get started.
You do not have to enable Delta Sharing as a consumer but you will have to make sure that you can access the provider storage (Keep in mind that data can be shared from a different cloud).

Accepting a share:

- You must be a metastore admin or have the USE PROVIDER privilege
- You can use the UI to accept the share and mount this as a catalog or the Delta Sharing REST API.
- Once the share is mounted to a catalog you'll manage it the same way you manage any other catalog

- The tables are now accessible similar to any other local table

You can also consume the data outside of your Databricks platform. See this documentation to get started

OAuth - ISV product integration best practices

Document available in folder
https://drive.google.com/drive/u/0/folders/1BMHK-xwLd1fSL38C-4Y53QBqYZBFcGdB

UC Volumes based staging locations - ISV product integration best practices

Document available in folder
https://drive.google.com/drive/u/0/folders/1BMHK-xwLd1fSL38C-4Y53QBqYZBFcGdB

UC personal staging locations [Deprecated] - ISV product integration best practices

Document available in folder
https://drive.google.com/drive/u/0/folders/1BMHK-xwLd1fSL38C-4Y53QBqYZBFcGdB

Reachout and get approval from Partner SA before using this feature

Governance & Observability

Document named UC best practices available in folder
https://drive.google.com/drive/u/0/folders/1BMHK-xwLd1fSL38C-4Y53QBqYZBFcGdB

Resources

Resources around Delta and Databricks SQL

Here are some useful links for integrating with Databricks using SQL

- Databricks Delta Documentation
  - https://docs.databricks.com/delta/index.html

- Delta Quickstart notebook
  - https://docs.databricks.com/delta/intro-notebooks.html#delta-lake-quickstart-sql-notebook

- SQL Reference
  - https://docs.databricks.com/sql/language-manual/index.html

- Copy Into command to ingest new data into Delta
  - https://docs.databricks.com/spark/latest/spark-sql/language-manual/delta-copy-into.html

- Merge command examples
  - https://docs.databricks.com/delta/delta-update.html#merge-examples
  - https://docs.databricks.com/delta/delta-update.html#upsert-into-a-table-using-merge
  - https://docs.databricks.com/delta/delta-update.html#!#write-change-data-into-a-delta-table

- Setting and retrieving table commit metadata
  - Use the following to query for the table history. userMetadata is one of the output attributes for history.
    - https://docs.databricks.com/delta/delta-utility.html#retrieve-delta-table-history
  - Details on Set user-defined commit metadata at
    - https://docs.databricks.com/delta/delta-batch.html#write-to-a-table

Quickstarts around Clusters

Cluster log delivery

When you create a cluster, you can specify a location to deliver Spark driver, worker, and event logs. Logs are delivered every five minutes to your chosen destination. When a cluster is terminated, Databricks guarantees to deliver all logs generated up until the cluster was terminated.

The destination of the logs depends on the cluster ID. If the specified destination is dbfs:/cluster-log-delivery, cluster logs for 0630-191345-leap375 are delivered to dbfs:/cluster-log-delivery/0630-191345-leap375

If you choose an S3 destination for cluster logs, you must configure the cluster with an instance profile that can access the bucket. This instance profile must have both the PutObject and PutObjectAcl permissions

Additional information

- Setting up cluster log delivery: https://docs.databricks.com/dev-tools/api/latest/examples.html#cluster-log-delivery-examples
- Cluster log delivery: https://docs.databricks.com/clusters/configure.html#cluster-log-delivery-1

Setting log level

Details on how to turn up log level for executors

https://kb.databricks.com/execution/set-executor-log-level.html#

To set the log level on all executors, set it inside the JVM on each worker.

Execute the code below to set it:

```
%scala
sc.parallelize(Seq("")).foreachPartition(x => {
  import org.apache.log4j.{LogManager, Level}
  import org.apache.commons.logging.LogFactory

  LogManager.getRootLogger().setLevel(Level.DEBUG)
  val log = LogFactory.getLog("EXECUTOR-LOG:")
  log.debug("START EXECUTOR DEBUG LOG LEVEL")
})
```

To verify that the level is set, navigate to the Spark UI, select the Executors tab, and open the stderr log for any executor.

To set the log level on a JVM

%scala

```scala
import org.apache.log4j.{LogManager, Level}
import org.apache.commons.logging.LogFactory

LogManager.getRootLogger().setLevel(Level.DEBUG)
val log = LogFactory.getLog("EXECUTOR-LOG:")
log.debug("START EXECUTOR DEBUG LOG LEVEL")
```

FAQ

1. Where to get Databricks Logo files?
    - Download the files from the Databricks brand portal at https://brand.databricks.com/databricks-brand-guidelines/visualsystem