databricks

Cloud Storage with LakeFlow Connect
Standard Connectors

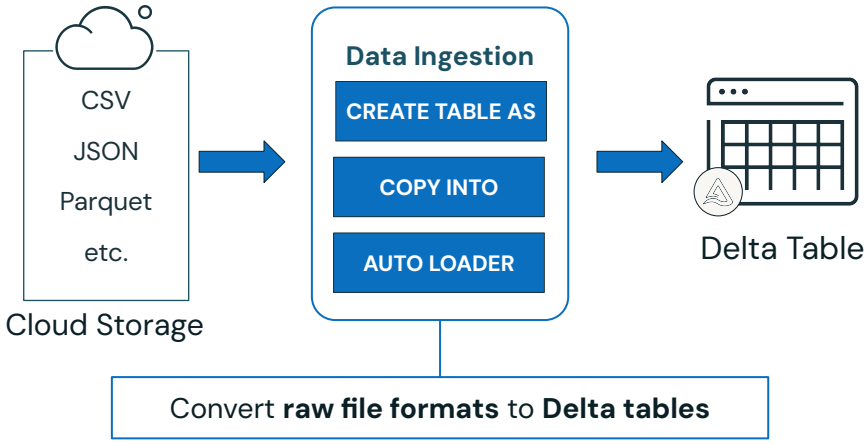
LECTURE

Introduction to Data Ingestion from Cloud Storage

In data ingestion from cloud storage, raw files are efficiently converted into Delta tables using Databricks tools—unlocking advanced management and analytics capabilities within the Lakehouse.

Data Ingestion from Cloud Storage

Data Ingestion Patterns From Cloud Object Storage



Data ingestion is a critical component of modern Lakehouse architecture, enabling organizations to take advantage of large volumes of data stored in cloud object storage systems. Common file formats like CSV, JSON, and Parquet are frequently used due to their flexibility and ease of use.

Our goal is to convert these raw files into Delta tables, unlocking advanced functionality such as ACID transactions, time travel, and schema enforcement.

We'll explore three primary methods for ingesting files from cloud object storage into Delta tables:

- CREATE TABLE AS (CTAS)
- COPY INTO
- Auto Loader

Ingestion from cloud object storage is performed using Lakeflow Connect Standard Connectors.

(1 of 3) CREATE TABLE AS (CTAS)

To begin, let's explore the primary method for ingesting files from cloud object storage into Delta tables using the CREATE TABLE AS approach.

Data Ingestion from Cloud Storage

Method 1 - Batch - CREATE TABLE AS (CTAS)


```
CREATE TABLE new_table AS
SELECT *
FROM read_files(
  <path_to_file(s)>,
  format => '<file_type>',
  <other_format_specific_options>
);
```

`CREATE TABLE AS (CTAS)` creates a Delta table **by default** from files in cloud object storage

The **`read_files()`** function reads files under a provided location and returns the data in **tabular form**

- Supports reading **file formats** like JSON, CSV, XML, TEXT, BINARYFILE, PARQUET, AVRO, and ORC file formats.
- Can detect the **file format automatically and infer a unified schema** across all files.
- Specify **specific file format options** to read in the data based on the source file format
- Can be used in **streaming tables** to **incrementally** ingest files into Delta Lake using Auto Loader

The CREATE TABLE AS (CTAS) statement creates a Delta table by default from files stored in cloud object storage.

The read_files() function is used to read files from a specified location and return the data in a tabular format. It offers several capabilities:

- Supports various file formats, including JSON, CSV, XML, TEXT, BINARYFILE, PARQUET, AVRO, and ORC.
- Automatically detects the file format and infers a unified schema across all files.
- Allows you to specify format-specific options for greater control when reading source files.
- Can be used in streaming tables to incrementally ingest files into Delta Lake using Auto Loader. We will learn more about Auto Loader shortly.

(2 of 3) COPY INTO

Next, let's examine the second primary method for ingesting files from cloud object storage into Delta tables using the COPY INTO command.

Data Ingestion from Cloud Storage

Method 2 - Incremental Batch - COPY INTO (legacy)


```
CREATE TABLE new_table;

COPY INTO new_table
FROM '<dir_path>'
FILEFORMAT=<file_type>
FORMAT_OPTIONS(<options>)
COPY_OPTIONS(<options>)
```

1. **Create an empty table to copy data into**
   - You can create an empty table **without** a schema
   - You also can **explicitly** create the table with a schema

Next, we will use the COPY INTO statement to copy files from cloud storage into the Delta table. This command performs a bulk load from files in cloud object storage into the table, and in this example, it will load files into the empty table new_table. The FROM clause specifies the location of the CSV files.

Typically, you start by creating table, which can be defined with or without a schema. In this case, we will create a table named new_table without a schema.

Data Ingestion from Cloud Storage

Method 2 - Incremental Batch - COPY INTO (legacy)


```
CREATE TABLE new_table;

COPY INTO new_table
FROM '<dir_path>'
FILEFORMAT=<file_type>
FORMAT_OPTIONS(<options>)
COPY_OPTIONS(<options>)
```

2. COPY INTO for incremental batch ingestion

- Is a **retriable and idempotent operation** and will **skip files** that have already been loaded **(incremental)**
- Supports various common files types like parquet, JSON, XML, etc
- **FROM** specifies the path of the cloud storage location continuously adding files
- **FORMAT_OPTIONS()** control how the source files are parsed and interpreted. The available options depends on the file format
- **COPY_OPTIONS()** controls the behavior of the COPY INTO operation itself, such as schema evolution (**mergeSchema**) and idempotency (**force**)

COPY INTO is ideal for situations where the cloud storage location is continuously adding files, since it is a retriable and idempotent operation designed for incremental batch ingestion.

What that means is: COPY INTO will skip any files that have already been loaded into the table, and only new files will be ingested.

Now, let's go over some of the key aspects of COPY INTO:

- It supports various common file types like Parquet, JSON, XML, and others.
- The FROM clause specifies the path of the cloud storage location where new files are being continuously added.
- FORMAT_OPTIONS() controls how the source files are parsed and interpreted, and the available options will depend on the file format you're working with.
- And finally, COPY_OPTIONS() lets you control the behavior of the COPY INTO operation itself. For example, options like schema evolution using mergeSchema, or idempotency using force.

(3 of 3) AUTO LOADER

Then, let's delve into the third major method for ingesting files from cloud object storage into Delta tables using Auto Loader.

Data Ingestion from Cloud Storage

Method 3 - Incremental Batch or Streaming - **Auto Loader**

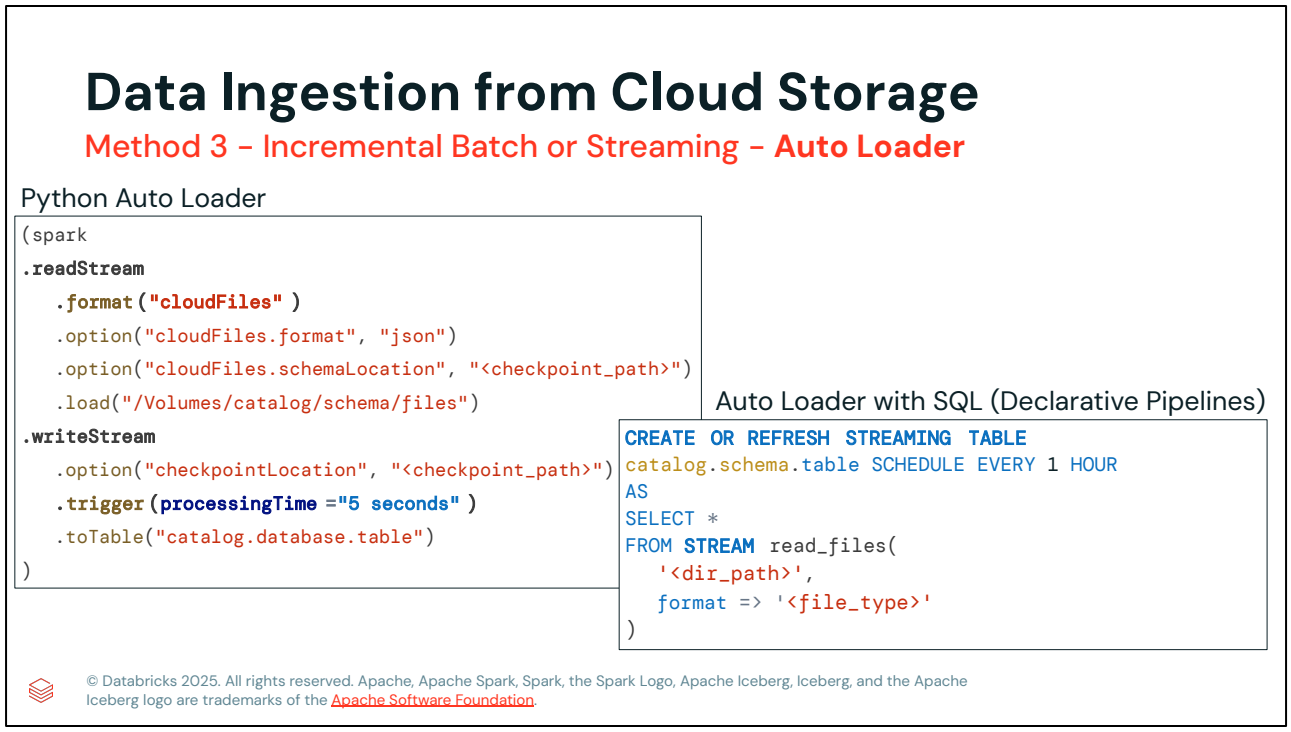
- **Incrementally** and **efficiently** processes new data files (in batch or streaming) as they arrive in cloud storage **without any additional setup**
- Auto Loader has support for both **Python** and **SQL (leveraging Declarative Pipelines)**
- You can use Auto Loader to process billions of files
- Auto Loader is built upon **Spark Structured Streaming**
- A deep dive into Auto Loader is **out of scope for this course**, please refer to these links and courses for more in depth information resources:
  - Documentation and Tutorials
    - Stream Processing and Analysis with Apache Spark™ Course
    - Build Data Pipelines with DLT Course
    - Databricks Streaming and DLT (Advanced) Course

Lastly, let's discuss incremental batch or streaming ingestion using Auto Loader.

- Auto Loader incrementally and efficiently processes new data files in either in batch or streaming mode as they arrive in cloud storage, and it does this without any additional setup.
- You can also use Auto Loader, which is a feature that automatically detects and ingests new files from cloud storage into Delta Lake tables, simplifying the process of handling incremental or streaming data with minimal configuration.
- Auto Loader has support for both Python and SQL, leveraging Declarative Pipelines.
- You can use Auto Loader to process billions of files, so it's built to scale.
- And finally, Auto Loader is built upon Spark Structured Streaming, which enables this efficient and reliable ingestion.

This course gives a high-level introduction into Auto Loader. For more information, please refer to these links and courses for more in-depth information and resources.

- Documentation and Tutorials
- Stream Processing and Analysis with Apache Spark™ Course
- Build Data Pipelines with DLT Course
- Databricks Streaming and DLT (Advanced) Course



Auto Loader in Python to read streaming data from cloud storage:

- We start with .readStream and set the format to "cloudFiles", which enables Auto Loader.
- Then, we specify the file format as JSON, and define the schema location using cloudFiles.schemaLocation, which is used to track schema inference and evolution.
- Next, we use .load() to point to the location of the files, in this case a path under /Volumes referencing Unity Catalog.
- On the write side, we configure .writeStream with a checkpoint location to maintain state and progress, and set a trigger interval of every 5 seconds.
- Finally, we use .toTable() to write the data into a Delta table specified by catalog, database, and table name.

Data Ingestion from Cloud Storage

Method 3 - Incremental Batch or Streaming - **Auto Loader**

Python Auto Loader

```
(spark
.readStream
   .format("cloudFiles" )
   .option("cloudFiles.format", "json")
   .option("cloudFiles.schemaLocation", "<checkpoint_path>")
   .load("/Volumes/catalog/schema/files")
.writeStream
   .option("checkpointLocation", "<checkpoint_path>")
   .trigger(processingTime ="5 seconds" )
   .toTable("catalog.database.table")
)
```

Auto Loader with SQL (Declarative Pipelines)

```
CREATE OR REFRESH STREAMING TABLE
catalog.schema.table SCHEDULE EVERY 1 HOUR
AS
SELECT *
FROM STREAM read_files(
   '<dir_path>',
   format => '<file_type>'
)
```


Auto Loader with Databricks SQL

- Databricks recommends using streaming tables to ingest data with Databricks SQL (instead of COPY INTO). A streaming table is a table registered to Unity Catalog that includes additional support for streaming or incremental data processing. When you create a streaming table, a pipeline is automatically generated for it. Streaming tables can be used for incremental data loading from both Kafka and cloud object storage.

- To create a streaming table from files in a volume, you use Auto Loader. Databricks recommends using Auto Loader with Lakeflow Declarative Pipelines for most data ingestion tasks from cloud object storage. Together, Auto Loader and Declarative Pipelines are designed to incrementally and idempotently load continuously growing datasets as they arrive.

- Streaming tables in Databricks SQL are backed by serverless Lakeflow Declarative Pipelines. Your workspace must support serverless pipelines to use this functionality. Alternatively, you can build your own Lakeflow Declarative Pipelines for incremental processing, optimization, and monitoring. Declarative Pipelines offer a range of additional features, which you can learn more about here: https://docs.databricks.com/aws/en/dlt/

- To use Auto Loader in Databricks SQL, use the read_files function with the STREAM keyword in the FROM clause.

Data Ingestion from Cloud Storage

FEATURE	CREATE TABLE AS (CTAS) + spark.read	COPY INTO	Auto Loader
Ingestion Type	Batch	Incremental Batch	Incremental (Batch or Streaming)
Use Cases	Best for smaller datasets	Ideal for thousands of files	Scale to millions+ of files per hour, backfills with billions of files
Syntax/Interface	•Python (spark.read) • SQL (CTAS)	SQL	• Python (spark.readStream) • SQL with Declarative Pipelines (CREATE OR REFRESH STREAMING TABLES) ○ Use streaming tables in Databricks SQL
Idempotency	No	Yes	Yes
Schema Evolution	Manual or inferred during read	Supported with options	Auto Loader automatically detects and evolves schemas. It supports loading data without predefined schemas and handles new columns as they appear.
Latency	High	Moderate (scheduled)	Low or high depending configuration
Easy of Use	Simple	Simple and SQL-based	Intermediate to advanced depending on the implementation (Python or SQL, incremental batch or streaming)
Summary	Best for one time, ad hoc ingestion. Can be scheduled to always read and process all data.	Simple and repeatable for incremental file ingestion. Great for schedule jobs or pipelines.	Best for near real-time streaming or incremental ingestion, with high automation and scalability.

Here's a quick summary of all three data ingestion methods.

Thank you for completing this lesson and continuing your journey to develop your skills with us.