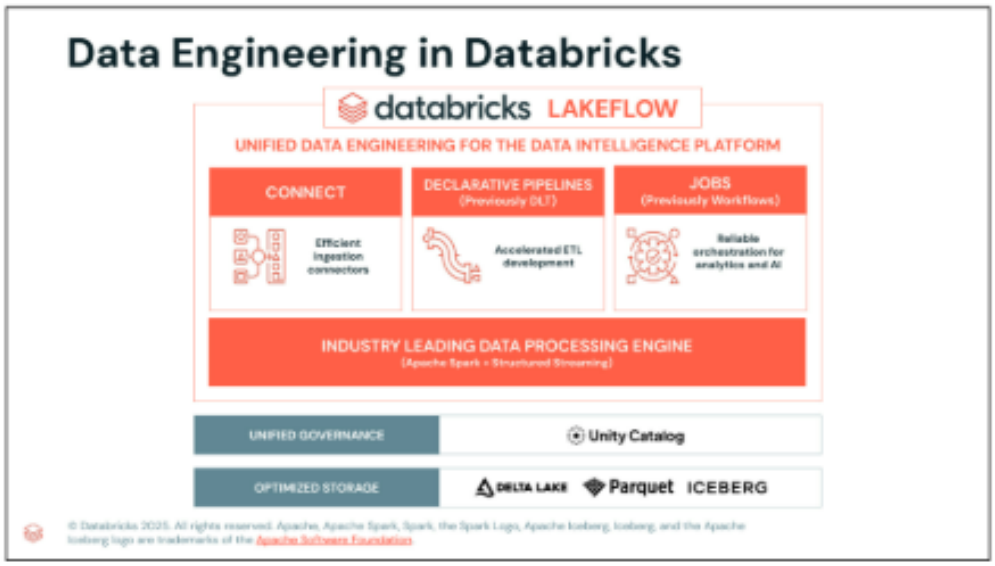
Data ingestion with lakeflow connect



It all begins with optimized storage using Delta Lake, Parquet, or Iceberg.

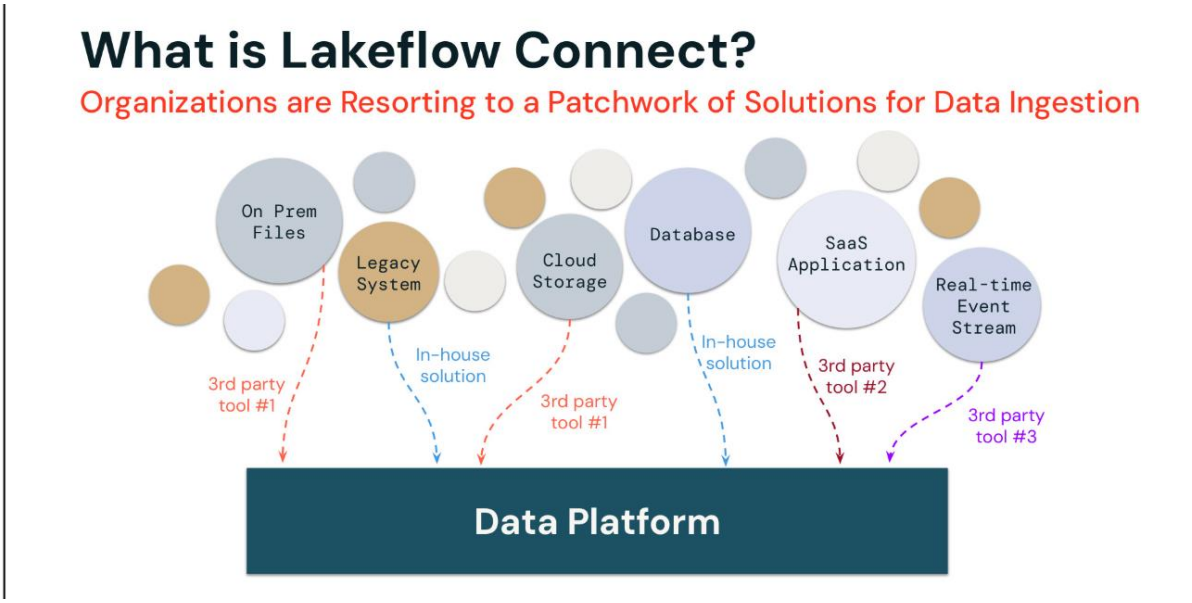
Built on top of this storage layer is unified governance with Unity Catalog. Unity Catalog is a centralized data catalog that provides access control, auditing, data lineage, quality monitoring, and data discovery across Databricks workspaces.

Databricks then offers **Lakeflow**, an end-to-end data engineering solution that empowers data engineers, software developers, SQL developers, analysts, and data scientists to deliver high-quality data for downstream analytics, AI, and operational applications. Lakeflow provides a unified platform for data ingestion, transformation, and orchestration, and includes the following components:

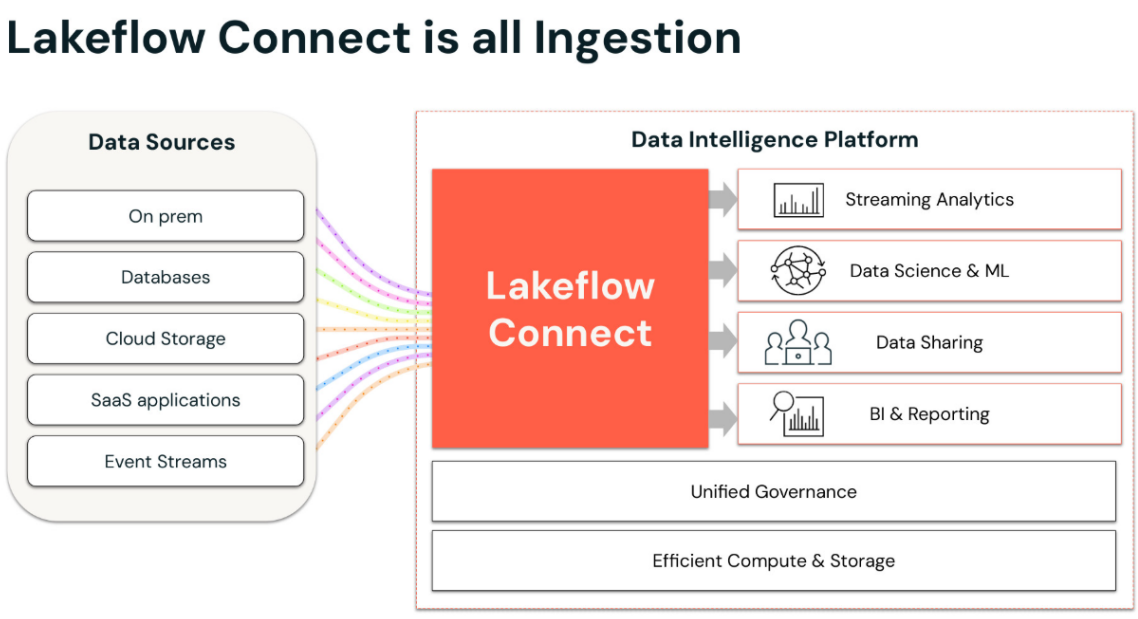
- **Lakeflow Connect**: A set of efficient ingestion connectors that simplify data ingestion from popular enterprise applications, databases, cloud storage, message buses, and local files.
- **Lakeflow Declarative Pipelines**: A framework for building batch and streaming data pipelines using SQL and Python, designed to accelerate ETL development.
- **Lakeflow Jobs**: A workflow automation tool for Databricks that orchestrates data processing workloads. It enables coordination of multiple tasks within complex workflows, allowing for the scheduling, optimization, and management of repeatable processes.

What is lakeflow connect?

In lakeflow connect, data ingestion is streamlined with simple, efficient connectors that enable you to bring in data from files, cloud storage, databases, enterprise applications, and streaming sources directly into the databricks lakehouse – all within a unified, managed platform.



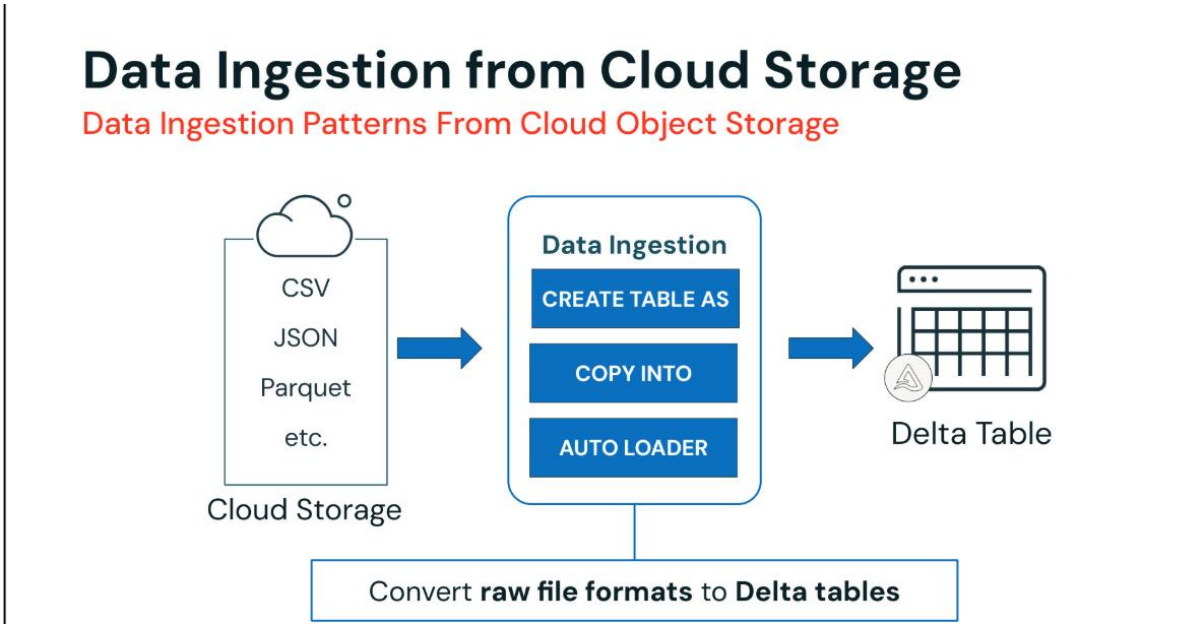
Traditionally, organizations are resorting to a patchwork of solution for data ingestion when working with enterprise systems, cloud storage and streaming.



With lakeflow connect, you can perform efficient ingestion pipeline all within databricks.

It's simple setup and maintenance, providing unified orchestration, observability, and governance all within the databricks data intelligence platform.

Data ingestion from cloud storage



Data ingestion is critical component of modern lakehous architecture, enabling organizations to take advantage of large volumes of data stored in cloud object storage systems.

Common file format like csv, json, parquet.

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

Three primary methods for ingesting file from cloud object storage into delta tables:

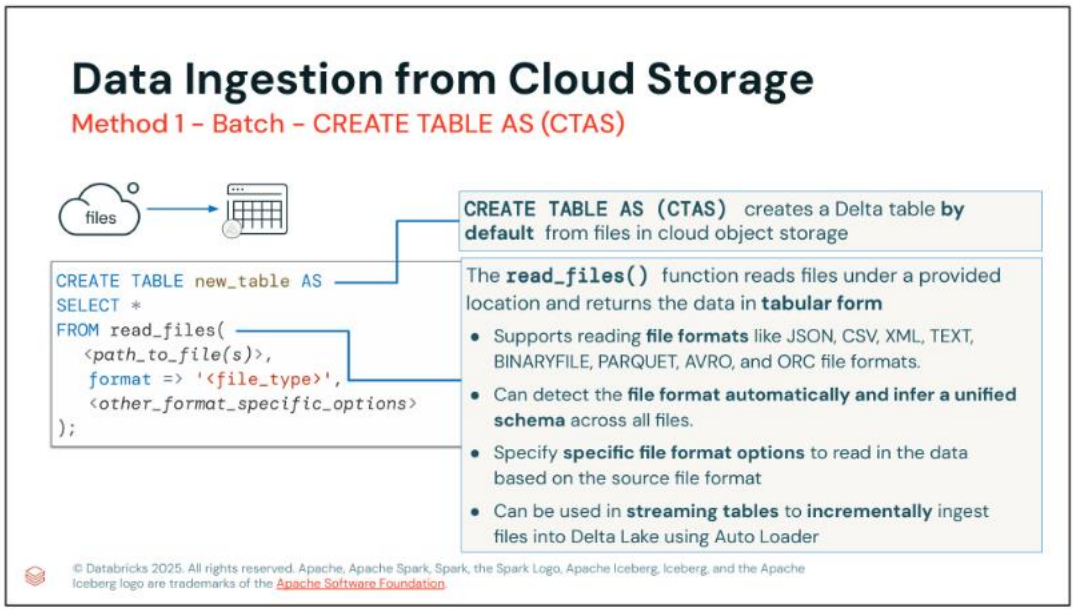
- Create table as
- Copy into
- Auto loader

Create table as (CTAS)

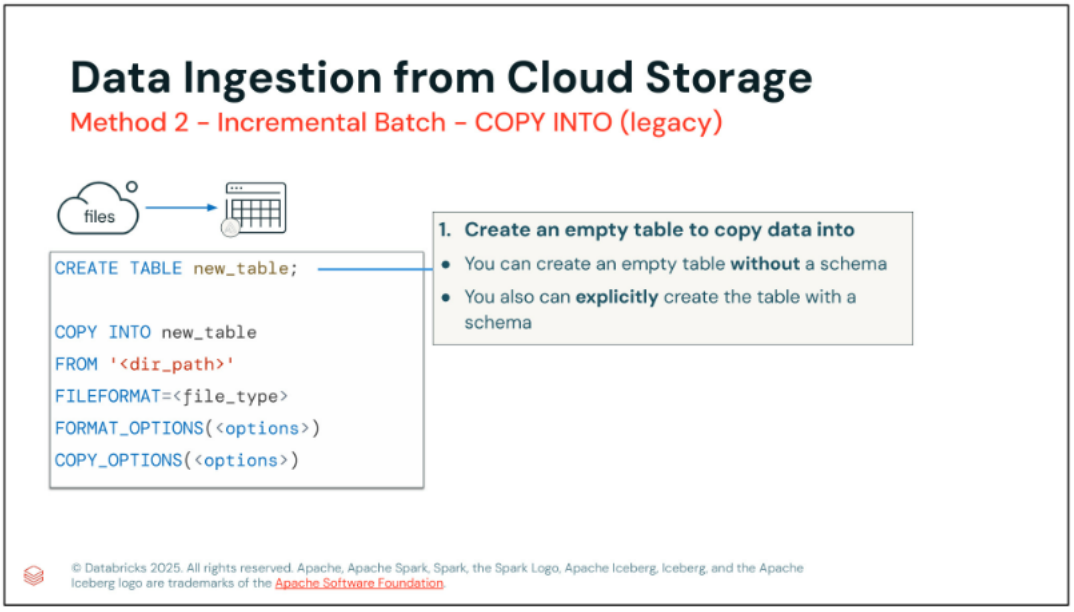
The CREATE TABLE AS (CTAS) statement creates a delta table by default from files stored in cloud object storage.

The read_file() 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 format, including json, csv, xml, text, binaryfile, parquet, avro, 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.

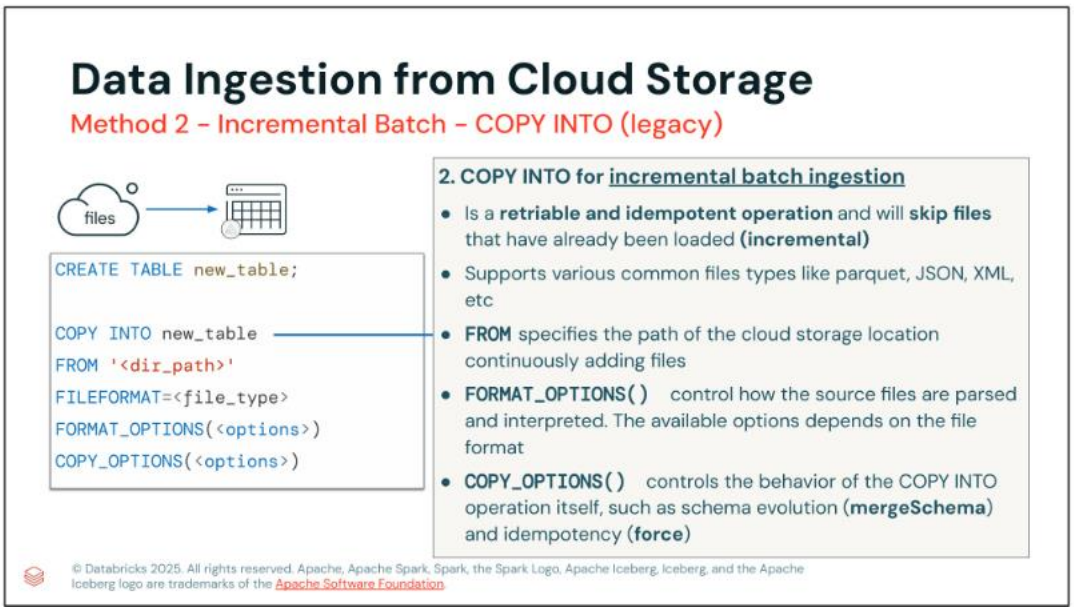


COPY INTO



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.

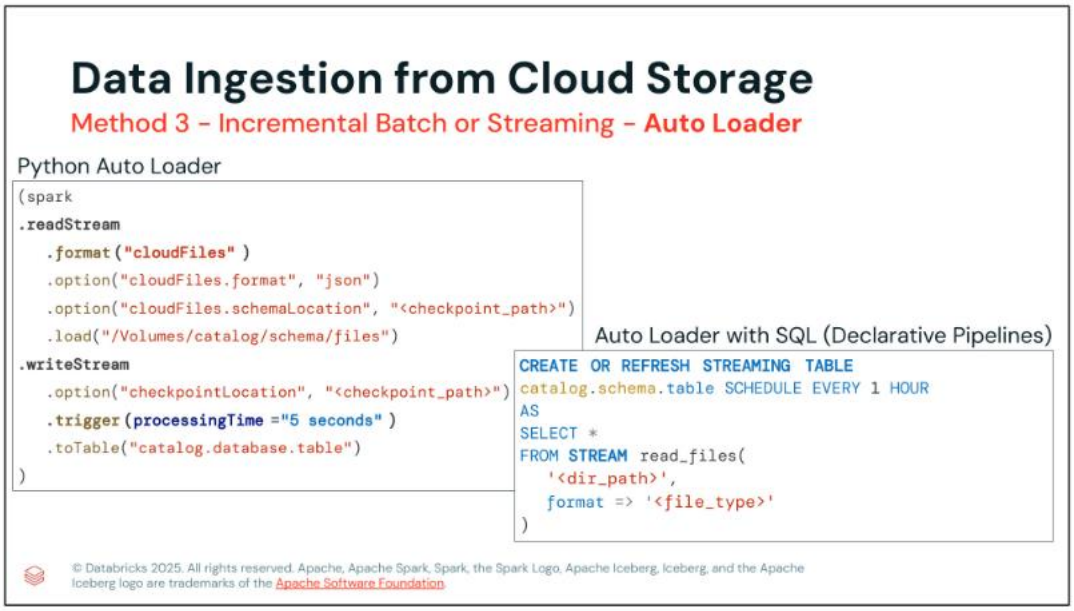


AUTO LOADER

Auto loader incrementally and efficiently process new data files in either in batch or streaming mode as they arrive in cloud storage, and it does this without any additional setup.

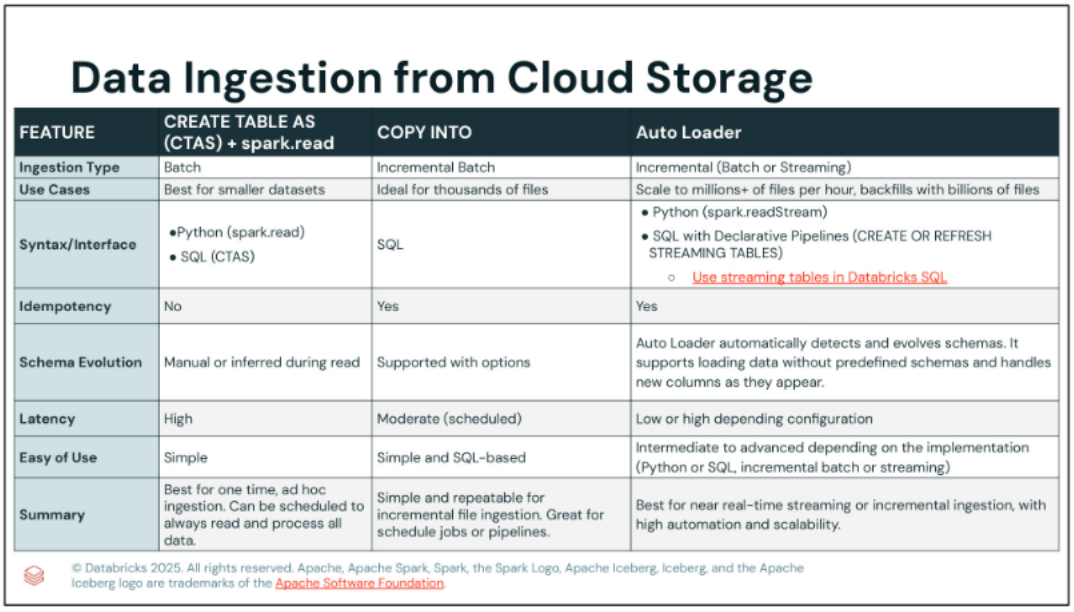
You can also use auto loader, which is a feature that automatically detects and ingests new file from cloud storage into delta lake tables, simplifying the process of handling incremental or streaming data with minimal configuration.

Auto loader built upon spark structured streaming. Which enables this efficient and reliable ingestion.



- We start with .**readStream** and set the format to **"cloudFiles",** which enables autoloader.
- Then, we specify the file format as JSON, and define the schema location using cloudFiles.schemeLocation, which is used to track schema inference and evolution.

- .load() to point to the location of files.
- .writeStream with a checkpoint location to maintain state and progress, and set a trigger interval of every 5 seconds.
- toTable() to write the data into a delta table



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.
