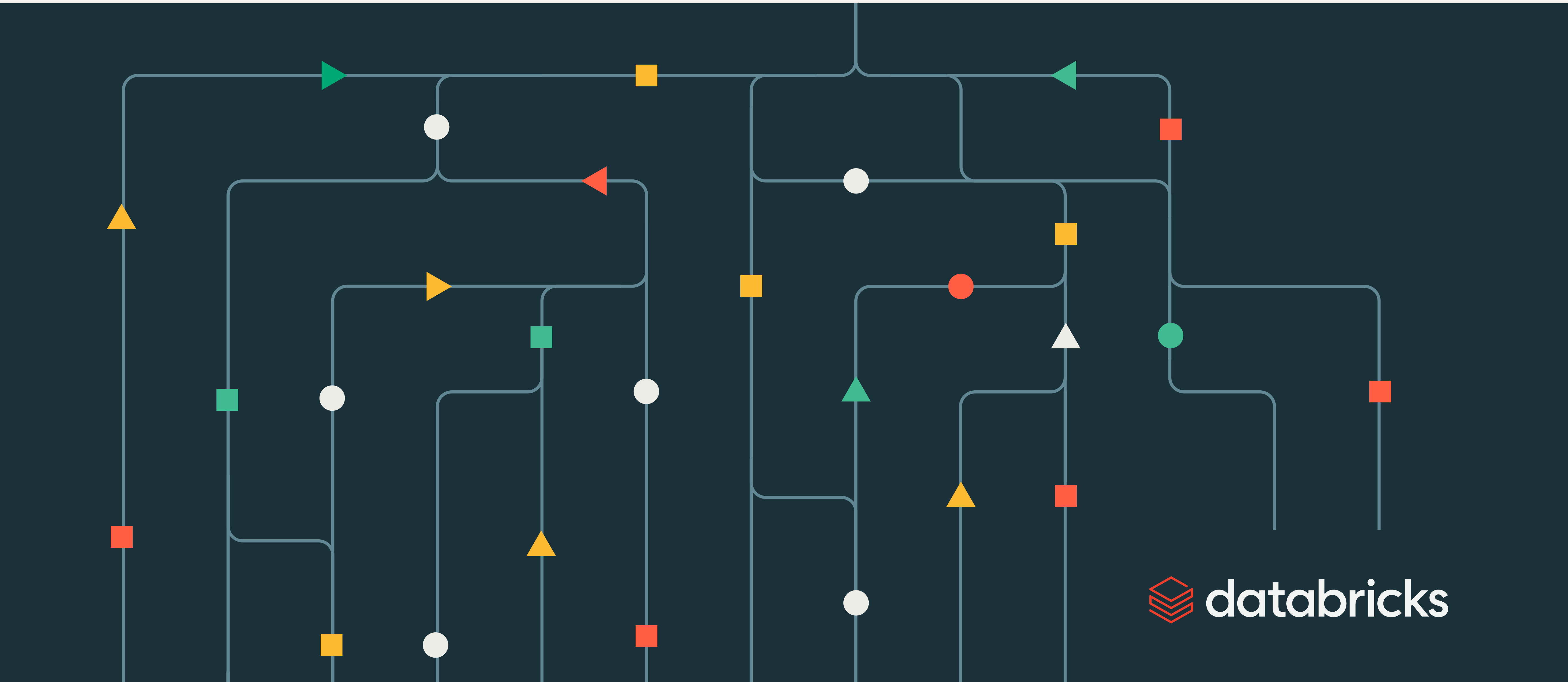
EBOOK

All Roads Lead to the Lakehouse

A deep dive into data ingestion with the lakehouse



databricks

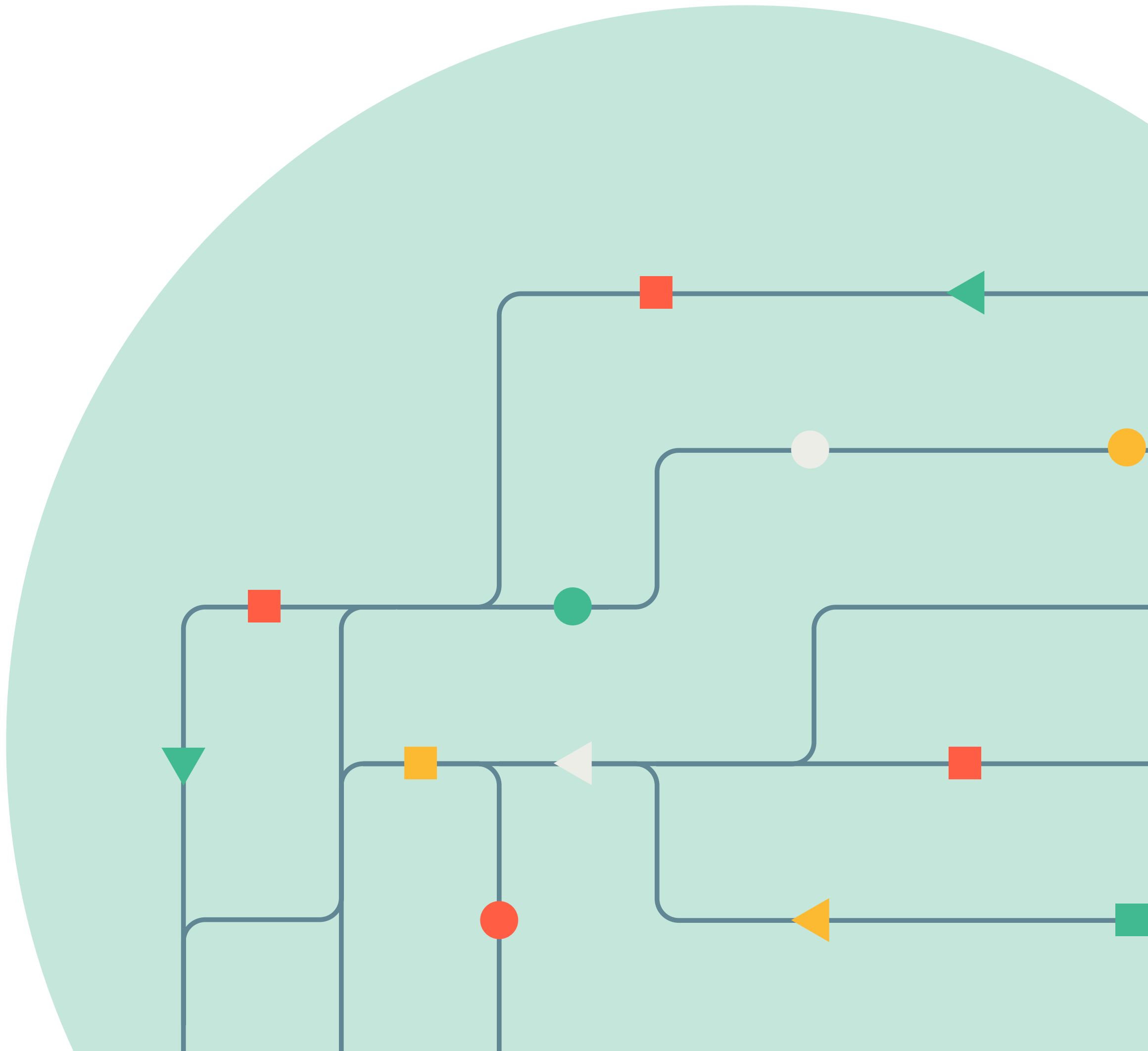
Contents

databricks

Introduction

Organizations today are inundated with data siloed across various on-premises application systems, databases, data warehouses and SaaS applications. This fragmentation makes it difficult to support new use cases for analytics or machine learning, so many IT teams are now centralizing all of their data with a lakehouse architecture built on top of Delta Lake, an open format storage layer.

The first thing data engineers need to do to support the lakehouse architecture is to efficiently move data from various systems into their lakehouse. Ingesting data is a critical first step in the data engineering and management lifecycle.



Life of a Data Engineer

The primary focus of data engineers is to provide timely and reliable data to downstream data teams at an organization. Requests for data can come from a variety of teams, and for a variety of data types. For example:

- Marketing team requests for Facebook and Google ad data in order to analyze spend and better allocate their budget for ads
- Security team looking to get access to a table with low latency security data from Kafka, in order to run rules to detect intrusions into the network
- Sales operations requesting customer data from Salesforce to enrich existing tables
- Finance team hoping to find a way to automatically ingest critical data from Google Sheets or transaction data from AWS Kinesis

In each of these common scenarios, data engineers must create usable and easily queryable tables from semi-structured and unstructured data. Beyond writing queries to retrieve and transform all this data, the data engineering team must also be concerned with performance, because running these queries on an ongoing basis can be a big load on the system.

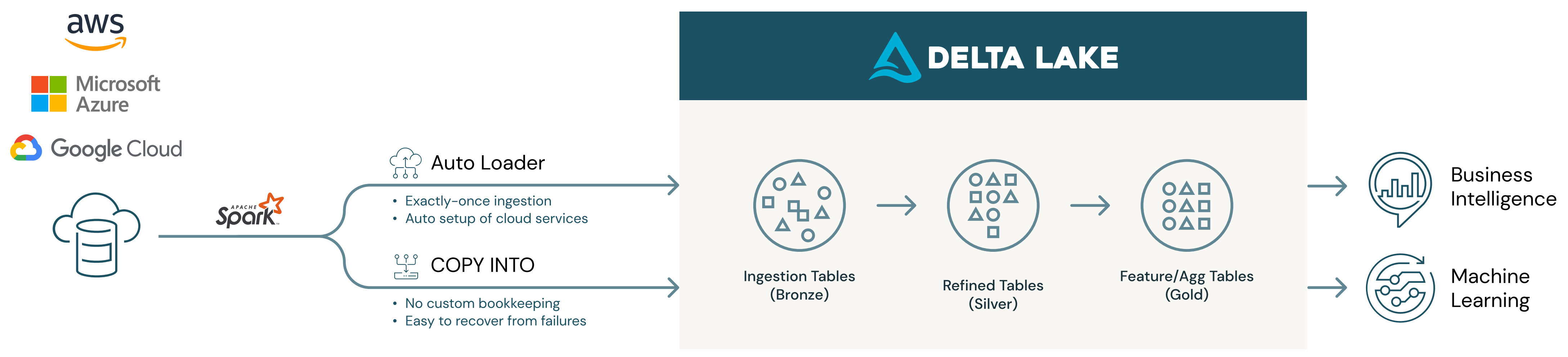
Data engineers face the challenge of constant requests and ongoing business requirements, as well as an ever-changing ecosystem. As business requirements change, so do the requirements around schemas, necessitating custom code to handle the changes. With all of these challenges, the work of a data engineer is extremely critical, and increasingly complex, with many steps involved before getting data to a state where it can actually be queried by the business stakeholders. So how do data engineers get the data that each of these teams need at the frequency, with the freshness, and in the format required?

WHAT IS DELTA LAKE?

Before thinking about ingestion into Delta Lake, it's important to understand why ingesting into Delta Lake is the right solution in the first place. Delta Lake is an open format data management layer that brings data warehouse capabilities to your open data lake. Across industries, enterprises have enabled true collaboration among their data teams with a reliable single source of truth enabled by Delta Lake. By delivering quality, reliability, security and performance on your data lake — for both streaming and batch operations — Delta Lake eliminates data silos and makes analytics accessible across the enterprise. With Delta Lake, customers can build a cost-efficient, highly scalable lakehouse that eliminates data silos and provides self-serving analytics to end users.

Ingesting From Cloud Object Stores

There are a number of common ways in which data engineers ingest data into Delta Lake. First and foremost is ingesting files from cloud object stores such as Azure Data Lake Storage, AWS S3 or Google Cloud Storage. Typically, customers are looking to migrate existing tables or perform incremental ingestion into Delta Lake, and to do so, they can leverage tools like CONVERT TO DELTA, COPY INTO, and Auto Loader. We will focus on Auto Loader and COPY INTO here.



Auto Loader
Auto Loader is an optimized data ingestion tool that incrementally and efficiently processes new data files as they arrive in cloud storage with minimal DevOps effort. You just need to provide a source directory path and start a streaming job. The new structured streaming source, called "cloudFiles", will automatically set up file notification services that subscribe file events from the input directory and process new files as they arrive, with the option of also processing existing files in that directory. Auto Loader has interfaces through Python and Scala, and can be used with SQL through Delta Live Tables.

COPY INTO
COPY INTO is a SQL command that allows you to perform batch file ingestion into Delta Lake. COPY INTO is a command that ingests files with exactly-once semantics, best used when the input directory contains thousands of files or fewer, and the user prefers SQL. COPY INTO can be used over JDBC to push data into Delta Lake at your convenience.

COPY INTO

COPY INTO is a powerful yet simple SQL command that allows you to perform batch file ingestion into Delta Lake and perform many of the use cases outlined in this section. COPY INTO can be run once, in an ad hoc manner, and can be scheduled through Databricks jobs. While COPY INTO does not support low latencies, you can trigger a COPY INTO based on events by using cloud functions such as AWS Lambda or through orchestrators like Apache Airflow. COPY INTO supports incremental appends and simple transformations.

COPY INTO is a great command to use when your source directory contains a small number of files (i.e., thousands of files or less). To ingest a larger number of files, we recommend Auto Loader, which we will cover later in this eBook.

Common Use Cases for COPY INTO

Ingesting data to a new Delta table

A common ad hoc ingestion use case using COPY INTO is to ingest data into a new Delta table. To copy data into a new Delta table, users can use CREATE TABLE command first, followed by COPY INTO.

```
Step 1: CREATE TABLE my_table (id INT, name STRING, age INT);
Step 2[1]: COPY INTO my_table
FROM 's3://my_bucket/my_path' WITH (
  CREDENTIAL (
    AWS_ACCESS_KEY = '*****',
    AWS_SECRET_KEY = '*****',
    AWS_SESSION_TOKEN = '*****'
  )
  ENCRYPTION (
    TYPE = 'AWS_SSE_C',
    MASTER_KEY = '*****'
  )
)
```

```
FILEFORMAT = CSV
FORMAT_OPTIONS ('header' = 'true')
```

The code block above covers the AWS temporary in-line credential format. When you use in-line credentials in Azure and AWS, the following parameters are required for each type of credential and encryption:

Credential Name	Required Parameters
AWS temporary credentials	AWS_ACCESS_KEY AWS_SECRET_KEY
	AWS_SESSION_TOKEN
Azure SAS token	AZURE_SAS_TOKEN

Encryption Name	Required Parameters
AWS server-side encryption with customer-provided encryption key	TYPE = 'AWS_SSE_C' MASTER_KEY
Azure client-provided encryption key	ATYPE = 'AZURE_CSE' MASTER_KEY

Appending data to your Delta table

To append data to a Delta table, users can leverage the COPY INTO command. COPY INTO is a powerful SQL command that is idempotent and incremental. When using COPY INTO, users point to a location of files, and once those files are ingested, Delta Lake will keep

[1] If you only have temporary access to a cloud object store, you can use temporary in-line credentials to ingest data from the cloud object store. When you are an admin or with ANY FILE access, and the instance profile has been set for the cloud object store, you do not need to specify credentials in-line for COPY INTO.

track of the state of files that have been ingested. Unlike commands like INSERT INTO, users get idempotency with COPY INTO, which means users are prevented from ingesting the same data twice to the same table.

```
COPY INTO table_identifier
FROM [ file_location | (SELECT expression_list FROM file_location)]
FILEFORMAT = JSON | CSV | TEXT | PARQUET | AVRO | ORC | BINARYFILE
[FILES = [file_name [,...] | PATTERN = 'regex_pattern']
[FORMAT_OPTIONS ('data_source_reader_option' = 'value' [, ...])]
[COPY_OPTIONS ('OPTION' = 'VALUE' [,...])]
```

One of the main benefits of COPY INTO is that users don't have to worry about providing a schema, because the schema is automatically inferred from your data files. Here is a very simple example of how you would ingest data from CSV files that have headers, where you leave the tool to infer the schema and the proper data types. It's as simple as that.

```
COPY INTO my_delta_table
FROM 's3://my-bucket/path/to/csv_files'
FILEFORMAT = CSV
FORMAT_OPTIONS ('header' = 'true', 'inferSchema' = 'true')
```

Using COPY INTO without an existing table[2]

In the most common case, in order to use COPY INTO, a table definition is required. However, if you would like to get started quickly and don't have an existing table or require a specific schema, you can create your table with a dummy schema. Then, once you run COPY INTO, you can overwrite the table and overwrite the schema. COPY INTO will actually infer the data types, and then change your Delta table to have the required schema.

```
CREATE TABLE my_delta_table (dummy string);
COPY INTO my_delta_table
FROM 's3://my-bucket/path/to/csv_files'
FILEFORMAT = CSV
FORMAT_OPTIONS (
 'header' = 'true',
 'inferSchema' = 'true',
 'mergeSchema' = 'true'
)
COPY_OPTIONS ( 'overwrite' = 'true', 'overwriteSchema' = 'true' )
```

Ingesting a CSV file without headers

If you are looking to ingest a CSV file that doesn't have headers, columns will be named as _c0 or _c1, with the index of the column. You can use the double colon syntax to cast the data type that you want and then alias these columns to whatever you want to call them.

```
COPY INTO my_delta_table
FROM ( SELECT
  _c0::int as key,
  _c1::double value,
  _c2::timestamp event_time
  FROM 's3://my-bucket/path/to/csv_files' )
FILEFORMAT = CSV
```

[2] This use case will not work in Databricks SQL workspace, as it currently only works on clusters without table ACLs.

Evolving schema over time for CSV files[3]

When ingesting CSV files that have a different number of columns than your existing table, you can use the option "'mergeSchema' = 'true'". This option needs to be provided both as FORMAT_OPTIONS and COPY_OPTIONS. FORMAT_OPTIONS applies to the source data. Once "mergeSchema" is provided as a format option, Databricks will look at multiple CSV files and infer the schema across those files. COPY_OPTIONS applies to your Delta table when you're running the COPY INTO command. When "mergeSchema" is provided as a copy option, you're instructing Delta Lake that it is safe to evolve the schema. Schema evolution only allows the addition of new columns. Data type changes for existing columns are not supported.

```
COPY INTO my_delta_table
FROM (SELECT
  _C0::int as key,
  _C1::double value,
  _C2::timestamp event_time,
  ...
  FROM 's3://my-bucket/path/to/csv_files')
FILEFORMAT = CSV
FORMAT_OPTIONS ('mergeSchema' = 'true')
COPY_OPTIONS ('mergeSchema' = 'true')
```

Fixing bad data

If you find that there is a mistake in the source data file and some of the data you ingested is bad, you can use RESTORE on your Delta table and set it to the timestamp or version of the Delta table that you want to roll back to (e.g., to restore to yesterday's data). Then you can rerun your COPY INTO command.

Alternatively, if running a RESTORE is not possible, COPY INTO supports reloading files by the use of the "force" copy option. You can manually remove the old data from your Delta Lake table by running a DELETE operation and then using COPY INTO with "force" = "true". You can use the PATTERN keyword to provide a file name pattern, or you can specify the file names with the FILES keyword to reload a subset of files in conjunction with "force".

```
RESTORE my_delta_table TO TIMESTAMP AS OF date_sub(current_date(),
1);
COPY INTO my_delta_table
FROM 's3://my-bucket/path/to/csv_files'
FILEFORMAT = CSV
PATTERN = '2021-09-08*.csv'
FORMAT_OPTIONS ( 'header' = 'true', 'inferSchema' = 'true' )
COPY_OPTIONS ('force' = 'true')
```

[3] Limitation: schema evolution with "mergeSchema" in COPY_OPTIONS does not work in Databricks SQL workspace or clusters enabled with table ACLs.

Auto Loader

While COPY INTO can solve a lot of the key use cases our customers face, due to its limitations (scalability), there are many scenarios where we recommend Auto Loader for data ingestion. Auto Loader is a data source on Databricks that incrementally and efficiently processes new data files as they arrive in cloud storage with minimal DevOps effort. Auto Loader is available in Python and Scala, and also in SQL in Delta Live Tables. Auto Loader is an incremental streaming source that provides exactly-once ingestion guarantees. It keeps track of which files have been ingested using a durable key-value store. It can discover new files very efficiently and is extremely scalable. Auto Loader has been battle tested. We have seen customers running Auto Loader on millions of files an hour, and petabytes of data per day.

To use Auto Loader, you simply specify 'readStream' and the format "cloudFiles", indicating that you will use Auto Loader to load files from the cloud object stores. Next, you specify the format of the file — for example, JSON — as an option to Auto Loader, and you specify where to load it from.

```
df = spark.readStream.format("cloudFiles")
  .option("cloudfiles.format", "json")
  .load("/path/to/table")
```

Under the hood, when data lands in your cloud storage, Auto Loader discovers files either through directory listing or file notifications. Given permissions to the underlying storage bucket or container, Auto Loader can list the directory that you want to load data from in an efficient and scalable manner and load data immediately. Alternatively, Auto Loader can also automatically set up file notifications on your storage account, which allows it to efficiently discover newly arriving files. When a file lands in file notification mode, the cloud storage system sends a notification to a queuing system. For example, in AWS, S3 will send a notification to AWS SQS. On Azure, a notification is sent to Azure queue storage. On Google, it'll be sent to Pub/Sub. Auto Loader can then fetch these event notifications from queues, deduplicate these notifications using its key-value store and then process the underlying files. If there are any failures, Auto Loader will replay what hasn't been processed, giving you exactly-once semantics.

Directory listing mode is very easy to get started with. If your files are uploaded to your cloud storage system in a lexicographical order, Auto Loader will optimize the discovery of files by starting directory listing from the latest uploaded files, saving you both time and money. If files cannot be uploaded in a lexicographical order and you need Auto Loader to scale to high volumes, Databricks recommends using the file notification mode. Cloud services such as AWS Kinesis Firehose, AWS DMS and Azure Data Factory can be configured to upload files in a lexical order, typically by providing the upload time of records in the file path, such as /base/path/yyyy/MM/dd/HH/file.format.

Common Use Cases for Auto Loader

New to Auto Loader

As a new user to the Databricks Lakehouse, you'll want to ingest data from cloud object stores into Delta Lake as part of your data pipeline for incremental loading. Here is a simple example using Python to demonstrate the ease and flexibility of Auto Loader with a few defined options. You can run the code in a notebook.

```
stream = spark.readStream \
  .format("cloudFiles") \
  .option("cloudFiles.format", "csv") \
  .option("cloudFiles.schemaLocation", schema_location) \
  .load(raw_data_location)
```

In order to write to a Delta table from the stream, follow the example below:

```
stream.writeStream \
   .option(“mergeSchema”, “true”) \
   .option(“checkpointLocation”, checkpoint_location) \
   .start(target_delta_table_location)
```

Migrating to Auto Loader

As a Spark user, you may be using an existing Spark structured streaming to process data. To migrate to Auto Loader, all a user needs to do is take existing streaming code and turn two lines of it into ‘cloudFiles’, specifying the file format within an option.

```
df = spark.readStream                   df = spark.readStream
  .format(“json”)                         .format(“cloudFiles”)
  .options(format_options)   ——————→      .option(“cloudFiles.
  .schema(schema)                       format”, “json”)
  .load(“/path/to/table”)                 .options(format_options)
                                          .schema(schema)
                                          .load(“/path/to/table”)
```

Once it’s converted, users will see instant benefits like scalability and cost reduction. Auto Loader can scale to trillions of files, unlike the open-source file streaming source. One of the ways that Auto Loader does this is with asynchronous backfills. Instead of needing to discover files first, then plan, Auto Loader discovers and processes files concurrently, making it much more efficient and leading to cost reductions in compute resources.

Migrating a livestreaming pipeline

Migrating a livestreaming pipeline can be challenging, but with Auto Loader, as with COPY INTO, you can specify a timestamp when the source files are updated or created and Auto Loader will ingest all modified data after that point.

```
df = spark.readStream
  .format(“cloudFiles”)
  .option(“cloudFiles.format”, “json”)
  .option(“modifiedAfter”, “2021-09-09 00:00:00”)
  .options(format_options)
  .schema(schema)
  .load(“/path/to/table”)
```

Schema inference and evolution

Auto Loader provides schema inference and management capabilities. With a schema location specified, Auto Loader can store the changes to the inferred schema over time. For file formats like JSON and CSV, where the schemas can get fuzzy, schema inference on Auto Loader can automatically infer data types or treat everything as a string.

When data does not match your schema (e.g., an unknown column or format), Auto Loader has a data rescue capability that will “rescue” all data in a separate column, stored as a JSON string, to investigate later. See rescued data column for more details.

Auto Loader supports three schema evolution modes: add new columns as they are discovered, fail if an unexpected column is seen, or rescue new columns.

Fixing a file that was processed with Auto Loader

To fix a file that was already processed, Auto Loader supports an option called 'allowOverwrites'. With this option, Auto Loader can re-ingest and reprocess a file with a new timestamp. If you want to enable this option in an existing Auto Loader stream, you need to stop and restart the Auto Loader stream with the enabled option.

```
df = spark.readStream
  .format("cloudFiles")
  .option("cloudFiles.format", "json")
  .schema(schema)
  .option("cloudFiles.allowOverwrites", "true")
  .options(format_options)
  .load("/path/to/table")
```

Discover missing data

While event notification is a very scalable method to collect all data, it relies on cloud services, which are distributed systems and are not always reliable. With Auto Loader, you can additionally specify a backfill interval, where Auto Loader will perform asynchronous backfills at whatever interval you set up. This can be enabled with a once trigger, processing time trigger and available now trigger. The following example shows how to use backfill internal and trigger availableNow together:

```
df = spark.readStream
  .format("cloudFiles")
  .option("cloudFiles.format", "json")
  .schema(schema)
  .option("cloudFiles.backfillInterval", "1 week")
  .options(format_options)
  .load("/path/to/table")
  .writeStream
  .trigger(Trigger.AvailableNow())
  .option("checkpointLocation", checkpointDir)
  .start()
```

The trigger tells Auto Loader how frequently to process incoming data. A processing time trigger will have Auto Loader run continuously and schedule micro-batches at the trigger interval which you have set. The "Once" and "AvailableNow" triggers instruct Auto Loader to process all new data that has been added until the start of your application. Once the data is processed, Auto Loader will automatically shut down. Trigger Once will have Auto Loader process all the new data in a single micro-batch, which requires it to first discover all the new files. With Trigger AvailableNow, Auto Loader can discover and process files concurrently and perform rate limiting, which makes it a preferable alternative to Trigger Once.

Using Auto Loader in SQL with Delta Live Tables

Delta Live Tables is a cloud-native ETL service on Databricks that provides a reliable framework to develop, test, monitor, manage and operationalize data pipelines at scale to drive insights for data science, machine learning and analytics. Auto Loader is available in Delta Live Tables.

```
CREATE INCREMENTAL LIVE TABLE
    autoloader_test
AS
SELECT
  *,
  id + id2 AS new_id
FROM
  CLOUD_FILES(
    “some/cloud/path”, - the path to the data
    “json” - the file format
  );
```

Live Tables understands and coordinates data flow between your queries

databricks

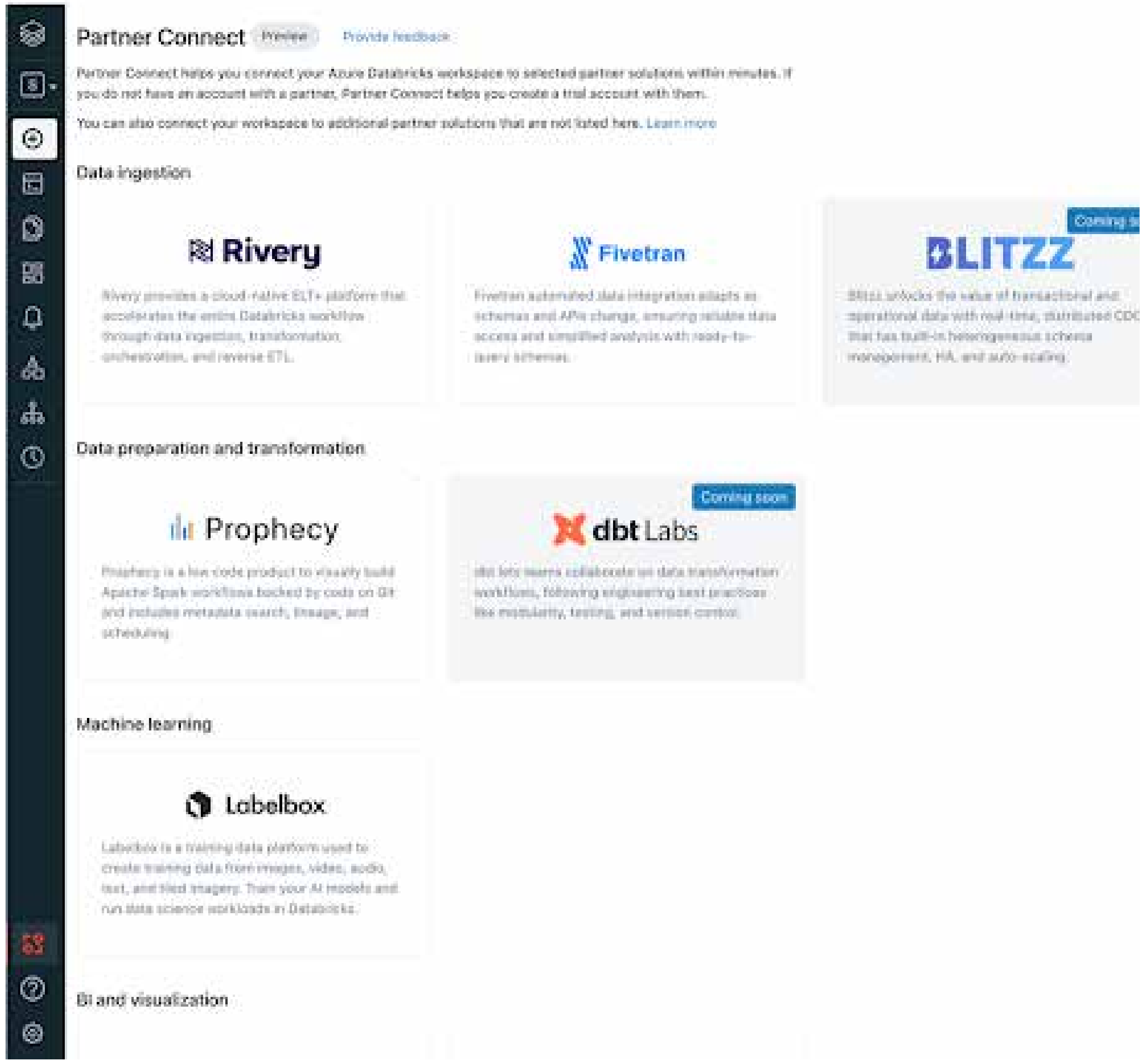
Ingesting Data From External Applications

While Auto Loader and COPY INTO are powerful tools, not all data is available as files in cloud object stores. In order to enable a lakehouse, it is critical to incorporate all of your data and break down the silos between sources and downstream teams. To do this, customers need to discover and connect a broad set of data, BI and AI tools, and systems to the data within their lakehouse.

Partner Connect

Historically, stitching multiple enterprise tools and data sources together has been a burden on the end user, making it very complicated and expensive to execute at any scale. Partner Connect solves this challenge by making it easy for you to integrate data, analytics and AI tools directly within their Databricks Lakehouse. It also allows you to discover new, pre-validated solutions from Databricks partners that support your expanding analytics needs.

To ingest into the lakehouse, select the partner tile in Partner Connect via the left navigation bar in Databricks. Partner Connect will automatically configure resources such as clusters, tokens and connection files for you to connect with your data ingestion tools of choice. You can finish signing up for a trial account on the partner's website or directly log in if you already used Partner Connect to create a trial account. Once you log in, you will see that Databricks is already configured as a destination in the partner portal and ready to be used.



Partner Connect helps you connect your Azure Databricks workspace to selected partner solutions within minutes. If you do not have an account with a partner, Partner Connect helps you create a trial account with them.

You can also connect your workspace to additional partner solutions that are not listed here. Learn more

Data ingestion

Rivery

Rivery provides a cloud-native ELT+ platform that accelerates the entire Databricks workflow through data ingestion, transformation, orchestration, and reverse ETL.

Fivetran

Fivetran automated data integration adapts as schemas and APIs change, ensuring reliable data access and simplified analysis with ready-to-query schemas.

Coming s

BLITZZ

Blitzz unlocks the value of transactional and operational data with real-time, distributed CDC that has built-in heterogeneous schema management, HA, and auto-scaling.

Data preparation and transformation

Prophecy

Prophecy is a low-code product to visually build Apache Spark workflows backed by code on Git and includes metadata search, lineage, and scheduling.

Coming soon

dbt Labs

dbt lets teams collaborate on data transformation workflows, following engineering best practices like modularity, testing, and version control.

Machine learning

Labelbox

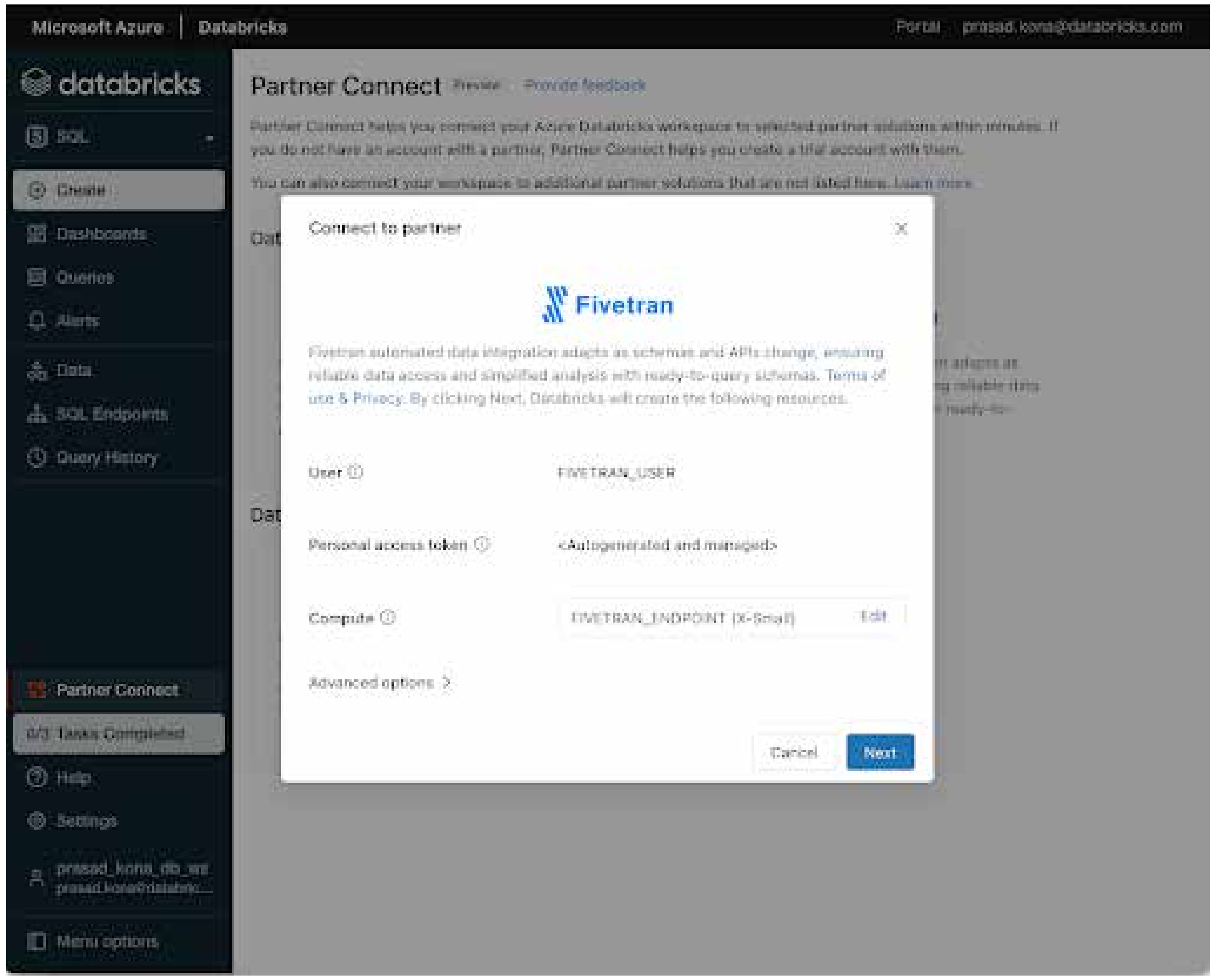
Labelbox is a training data platform used to create training data from images, video, audio, text, and tiled imagery. Train your AI models and run data science workloads in Databricks.

BI and visualization

databricks

Common Use Case for Partner Connect

Ingest Salesforce data via Fivetran into Delta Lake



Partner Connect

Partner Connect helps you connect your Azure Databricks workspace to selected partner solutions within minutes. If you do not have an account with a partner, Partner Connect helps you create a trial account with them.

You can also connect your workspace to additional partner solutions that are not listed here. Learn more

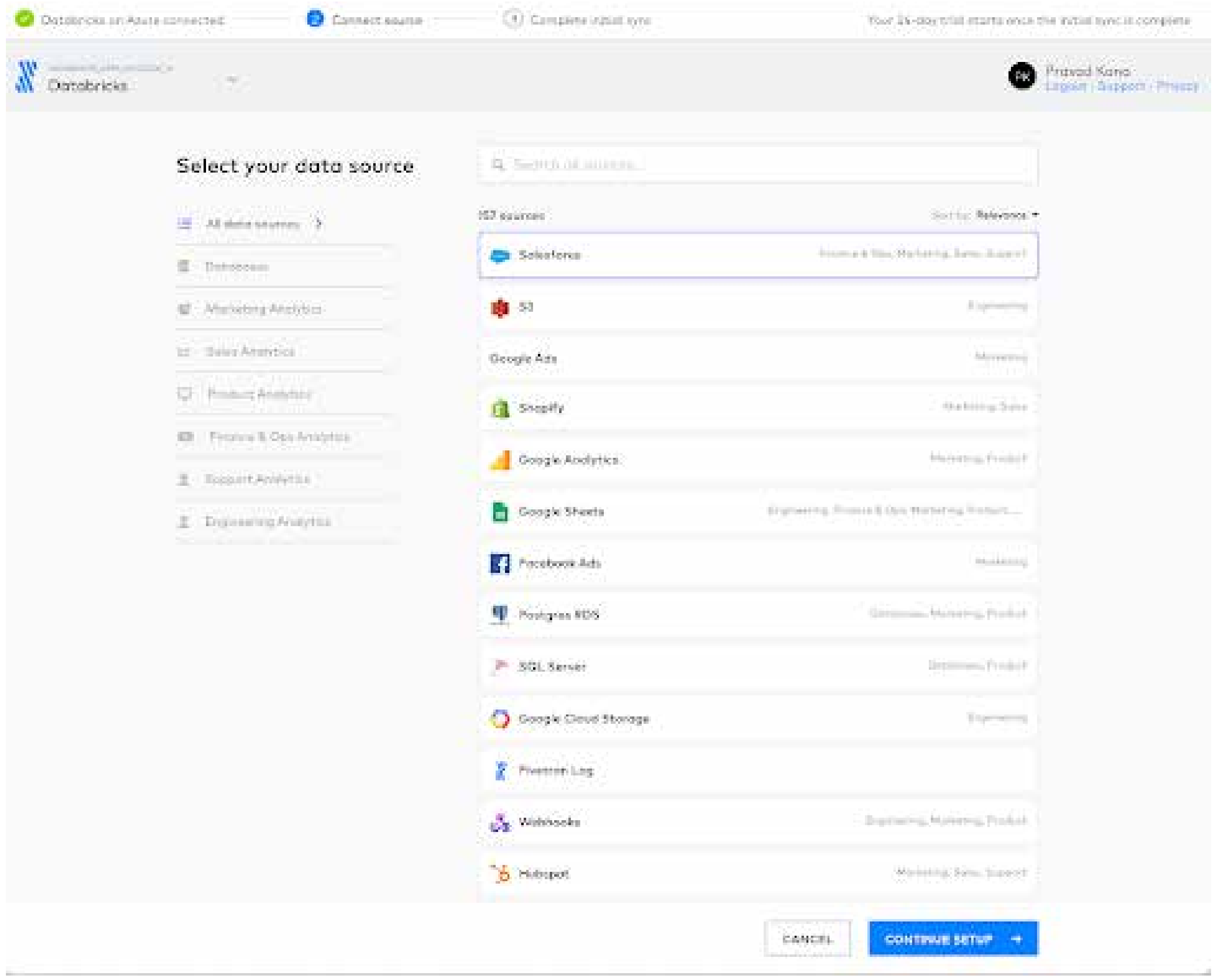
Connect to partner

Fivetran automated data integration adapts as schemas and APIs change, ensuring reliable data access and simplified analysis with ready-to-query schemas. Terms of use & Privacy. By clicking Next, Databricks will create the following resources.

User	FIVETRAN_USER
Personal access token	<Autogenerated and managed>
Compute	FIVETRAN_ENDPOINT (X-Small) Edit

Advanced options >

Clicking on the Fivetran tile in Partner Connect starts an automated workflow between the two products. Databricks automatically provisions a SQL endpoint and associated credentials for Fivetran to interact with, and passes the user's identity and the SQL endpoint configuration to Fivetran automatically via a secure API. Within Fivetran, a Databricks destination is automatically created. This destination is configured to ingest into Delta via the SQL endpoint that was auto-configured by Partner Connect.



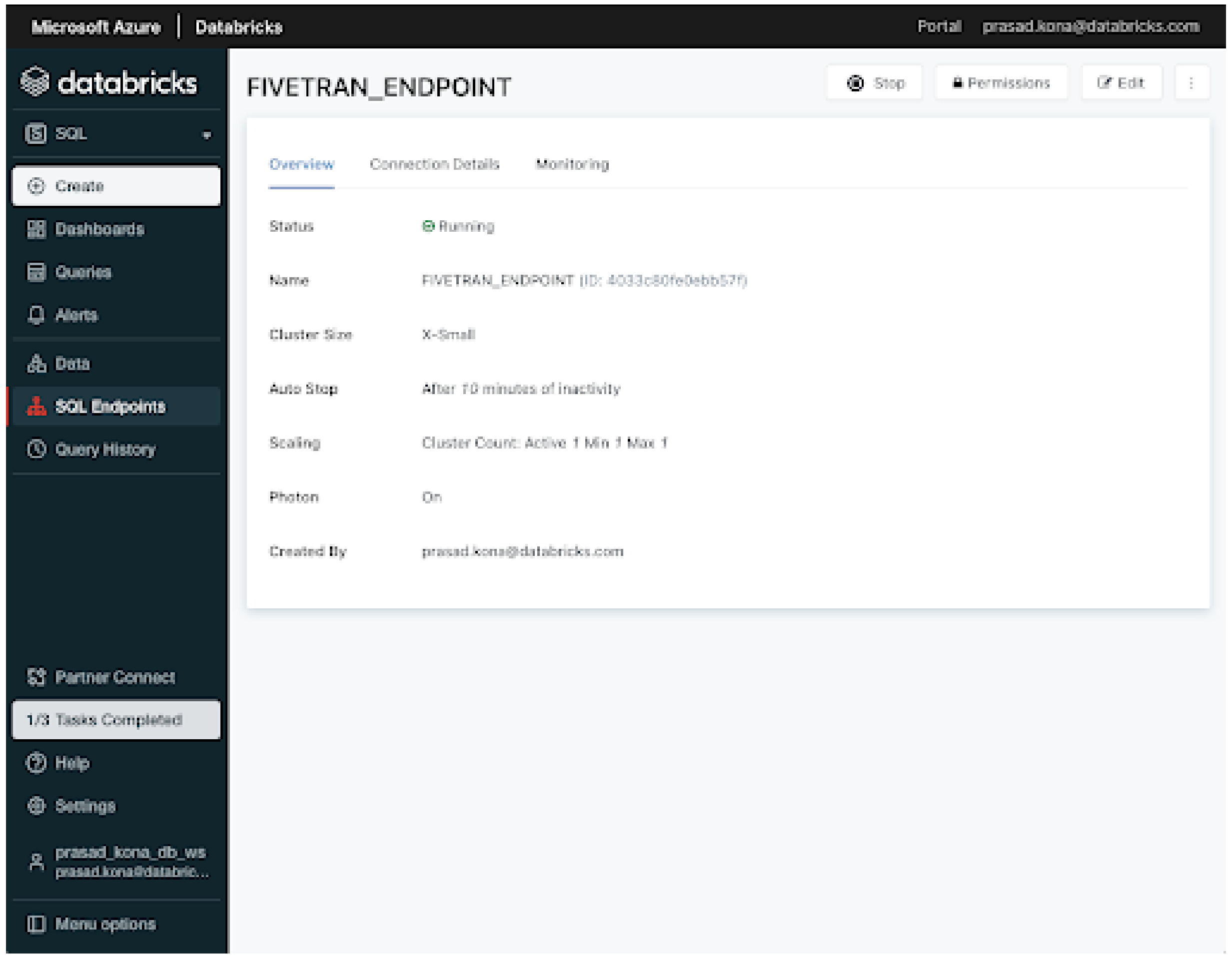
Your 14-day trial starts once the initial sync is complete

Select your data source

157 sources

Salesforce	Finance & Ops, Marketing, Sales, Support
S3	Engineering
Google Ads	Marketing
Shopify	Marketing, Sales
Google Analytics	Marketing, Product
Google Sheets	Engineering, Finance & Ops, Marketing, Product…
Facebook Ads	Marketing
Postgres RDS	Databases, Marketing, Product
SQL Server	Databases, Product
Google Cloud Storage	Engineering
Fivetran Log	
Webhooks	Engineering, Marketing, Product
Hubspot	Marketing, Sales, Support

The customer now selects their choice of data source in Fivetran from hundreds of pre-built connectors — for example, Salesforce. The user authenticates to the Salesforce source, chooses the Salesforce objects they want to ingest into Delta Lake on Databricks (in this case the Account & Contact objects) and starts the initial sync. This automation has saved users dozens of manual steps and copying/pasting of configuration if they manually set up the connection. It also protects the user from making any unintentional configuration errors and spending time debugging those errors. The Salesforce tables are now available to query, join and analyze in Databricks SQL. Watch the demo for more details or check out the Partner Connect guide to learn more.



FIVETRAN_ENDPOINT

Status	Running
Name	FIVETRAN_ENDPOINT (ID: 4033c80fe0ebb57f)
Cluster Size	X-Small
Auto Stop	After 10 minutes of inactivity
Scaling	Cluster Count: Active 1 Min 1 Max 1
Photon	On
Created By	prasad.kona@databricks.com

About Databricks

Databricks is the data and AI company. More than 5,000 organizations worldwide — including Comcast, Condé Nast, H&M and over 40% of the Fortune 500 — rely on the Databricks Lakehouse Platform to unify their data, analytics and AI. Databricks is headquartered in San Francisco, with offices around the globe. Founded by the original creators of Apache Spark™, Delta Lake and MLflow, Databricks is on a mission to help data teams solve the world's toughest problems. To learn more, follow Databricks on Twitter LinkedIn and Facebook

databricks