databricks

EBOOK

# The Big Book of Data Engineering 2nd Edition

A collection of technical blogs, including code samples and notebooks

With all-new content

# Contents

SECTION

01

# Introduction to Data Engineering on Databricks

Organizations realize the value data plays as a strategic asset for various business-related initiatives, such as growing revenues, improving the customer experience, operating efficiently or improving a product or service. However, accessing and managing data for these initiatives has become increasingly complex. Most of the complexity has arisen with the explosion of data volumes and data types, with organizations amassing an estimated 80% of data in unstructured and semi-structured format. As the collection of data continues to increase, 73% of the data goes unused for analytics or decision-making. In order to try and decrease this percentage and make more data usable, data engineering teams are responsible for building data pipelines to efficiently and reliably deliver data. But the process of building these complex data pipelines comes with a number of difficulties:

- In order to get data into a data lake, data engineers are required to spend immense time hand-coding repetitive data ingestion tasks
- Since data platforms continuously change, data engineers spend time building and maintaining, and then rebuilding, complex scalable infrastructure
- As data pipelines become more complex, data engineers are required to find reliable tools to orchestrate these pipelines
- With the increasing importance of real-time data, low latency data pipelines are required, which are even more difficult to build and maintain
- Finally, with all pipelines written, data engineers need to constantly focus on performance, tuning pipelines and architectures to meet SLAs

## How can Databricks help?

With the Databricks Lakehouse Platform, data engineers have access to an end-to-end data engineering solution for ingesting, transforming, processing, scheduling and delivering data. The Lakehouse Platform automates the complexity of building and maintaining pipelines and running ETL workloads directly on a data lake so data engineers can focus on quality and reliability to drive valuable insights.

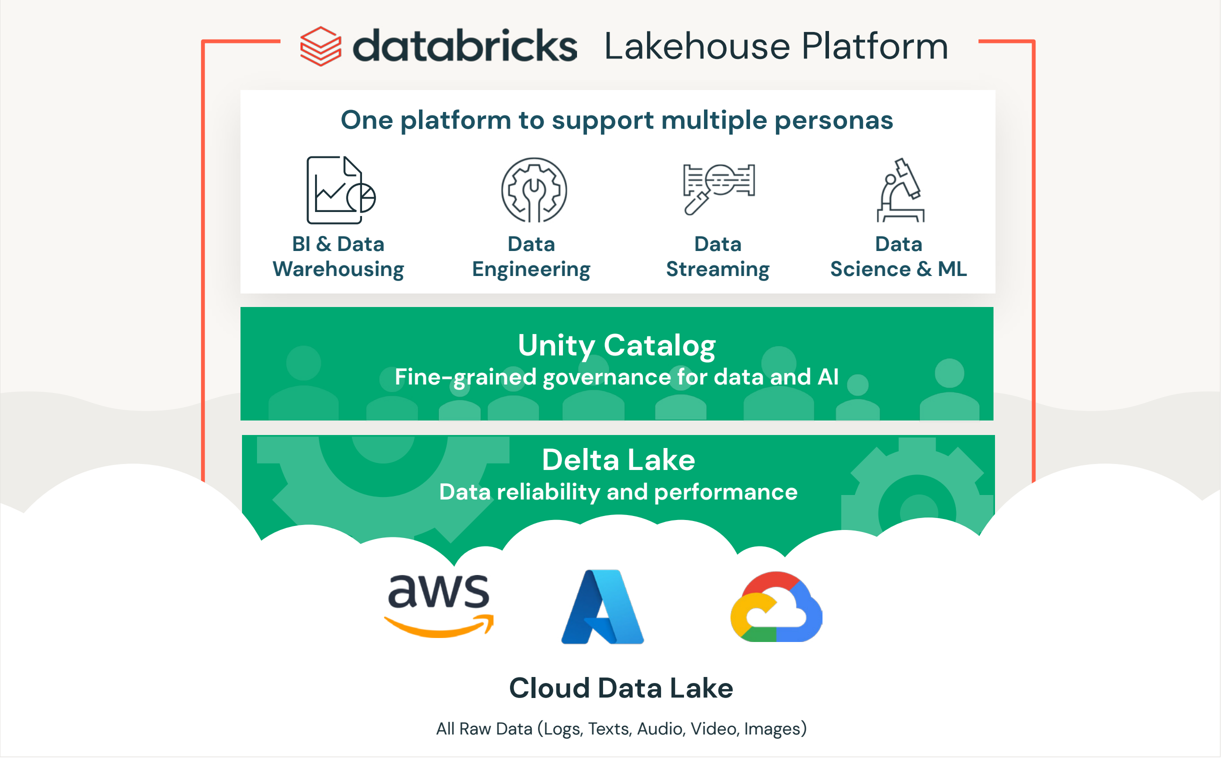

databricks Lakehouse Platform
One platform to support multiple personas
BI & Data Warehousing
Data Engineering
Data Streaming
Data Science & ML
Unity Catalog
Fine-grained governance for data and AI
Delta Lake
Data reliability and performance
aws
Cloud Data Lake
All Raw Data (Logs, Texts, Audio, Video, Images)

Figure 1
The Databricks Lakehouse Platform unifies your data, analytics and AI on one common platform for all your data use cases

## Key differentiators for successful data engineering with Databricks

By simplifying on a lakehouse architecture, data engineers need an enterprise-grade and enterprise-ready approach to building data pipelines. To be successful, a data engineering solution team must embrace these eight key differentiating capabilities:

### Data ingestion at scale

With the ability to ingest petabytes of data with auto-evolving schemas, data engineers can deliver fast, reliable, scalable and automatic data for analytics, data science or machine learning. This includes:

- Incrementally and efficiently processing data as it arrives from files or streaming sources like Kafka, DBMS and NoSQL
- Automatically inferring schema and detecting column changes for structured and unstructured data formats
- Automatically and efficiently tracking data as it arrives with no manual intervention
- Preventing data loss by rescuing data columns

### Declarative ETL pipelines

Data engineers can reduce development time and effort and instead focus on implementing business logic and data quality checks within the data pipeline using SQL or Python. This can be achieved by:

- Using intent-driven declarative development to simplify "how" and define "what" to solve
- Automatically creating high-quality lineage and managing table dependencies across the data pipeline
- Automatically checking for missing dependencies or syntax errors, and managing data pipeline recovery

### Real-time data processing

Allow data engineers to tune data latency with cost controls without the need to know complex stream processing or implement recovery logic.

- Avoid handling batch and real-time streaming data sources separately
- Execute data pipeline workloads on automatically provisioned elastic Apache Spark™-based compute clusters for scale and performance
- Remove the need to manage infrastructure and focus on the business logic for downstream use cases

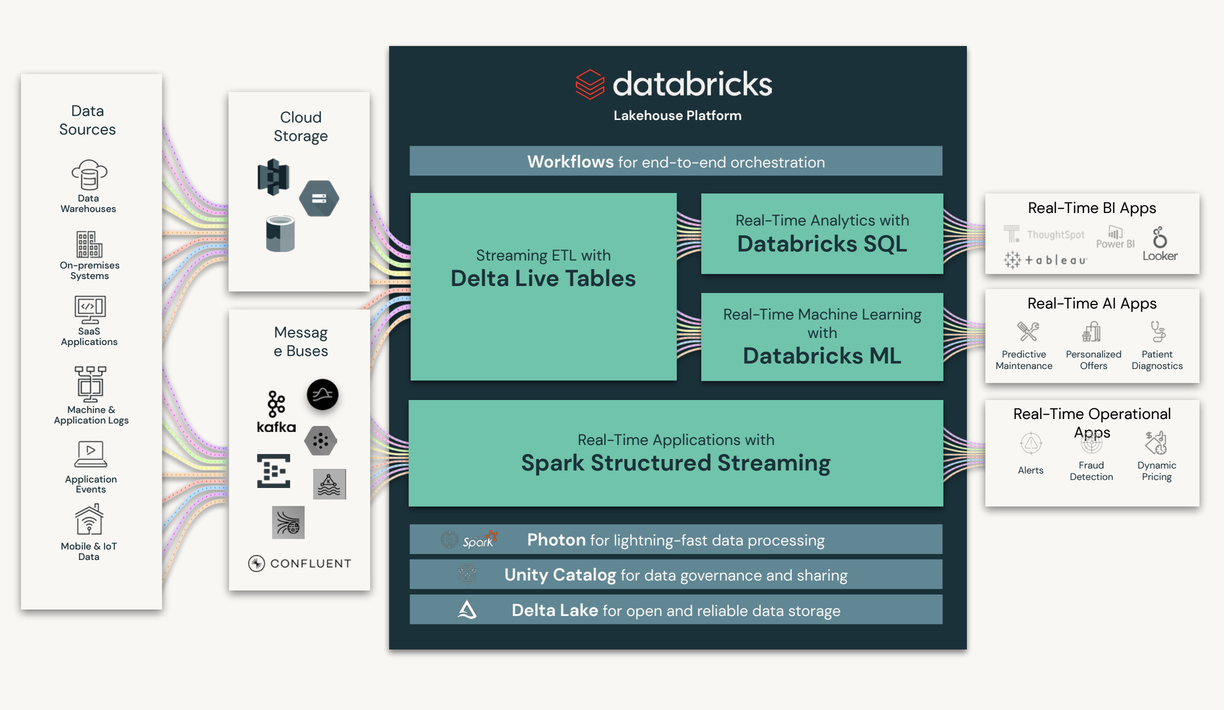


Figure 2
A unified set of tools for real-time data processing

## Unified orchestration of data workflows

Simple, clear and reliable orchestration of data processing tasks for data, analytics and machine learning pipelines with the ability to run multiple non-interactive tasks  as a directed acyclic graph (DAG) on a Databricks compute cluster. Orchestrate tasks of any kind (SQL, Python, JARs, Notebooks) in a DAG using Databricks Workflows, an orchestration tool included in the lakehouse with no need to maintain or pay for an external orchestration service.

- Easily create and manage multiple tasks with dependencies via UI, API or from your IDE
- Have full observability to all workflow runs and get alerted when tasks fail for fast troubleshooting and efficient repair and rerun
- Leverage high reliability of 99.95% uptime
- Use performance optimization clusters that parallelize jobs and minimize data movement with cluster reuse

## Data quality validation and monitoring

Improve data reliability throughout the data lakehouse so data teams can confidently trust the information for downstream initiatives by:

- Defining data quality and integrity controls within the pipeline with defined data expectations
- Addressing data quality errors with predefined policies (fail, drop, alert, quarantine)
- Leveraging the data quality metrics that are captured, tracked and reported for the entire data pipeline

## Fault tolerant and automatic recovery

Handle transient errors and recover from most common error conditions occurring during the operation of a pipeline with fast, scalable automatic recovery that includes:

- Fault tolerant mechanisms to consistently recover the state of data
- The ability to automatically track progress from the source with checkpointing
- The ability to automatically recover and restore the data pipeline state

## Data pipeline observability

Monitor overall data pipeline status from a dataflow graph dashboard and visually track end-to-end pipeline health for performance, quality and latency. Data pipeline observability capabilities include:

- A high-quality, high-fidelity lineage diagram that provides visibility into how data flows for impact analysis
- Granular logging with performance and status of the data pipeline at a row level
- Continuous monitoring of data pipeline jobs to ensure continued operation

## Automatic deployments and operations

Ensure reliable and predictable delivery of data for analytics and machine learning use cases by enabling easy and automatic data pipeline deployments and rollbacks to minimize downtime. Benefits include:

- Complete, parameterized and automated deployment for the continuous delivery of data
- End-to-end orchestration, testing and monitoring of data pipeline deployment across all major cloud providers

## Migrations

Accelerating and de-risking the migration journey to the lakehouse, whether from legacy on-prem systems or disparate cloud services.

The migration process starts with a detailed discovery and assessment to get insights on legacy platform workloads and estimate migration as well as Databricks platform consumption costs. Get help with the target architecture and how the current technology stack maps to Databricks, followed by a phased implementation based on priorities and business needs. Throughout this journey companies can leverage:

- Automation tools from Databricks and its ISV partners
- Global and/or regional SIs who have created Brickbuilder migration solutions
- Databricks Professional Services and training

This is the recommended approach for a successful migration, whereby customers have seen a 25-50% reduction in costs and 2-3x faster time to value for their use cases.

## Unified governance

With Unity Catalog, data engineering and governance teams benefit from an enterprisewide data catalog with a single interface to manage permissions, centralize auditing, automatically track data lineage down to the column level, and share data across platforms, clouds and regions. Benefits:

- Discover all your data in one place, no matter where it lives, and centrally manage fine-grained access permissions using an ANSI SQL-based interface
- Leverage automated column-level data lineage to perform impact analysis of any data changes across the pipeline and conduct root cause analysis of any errors in the data pipelines
- Centrally audit data entitlements and access
- Share data across clouds, regions and data platforms, while maintaining a single copy of your data in your cloud storage

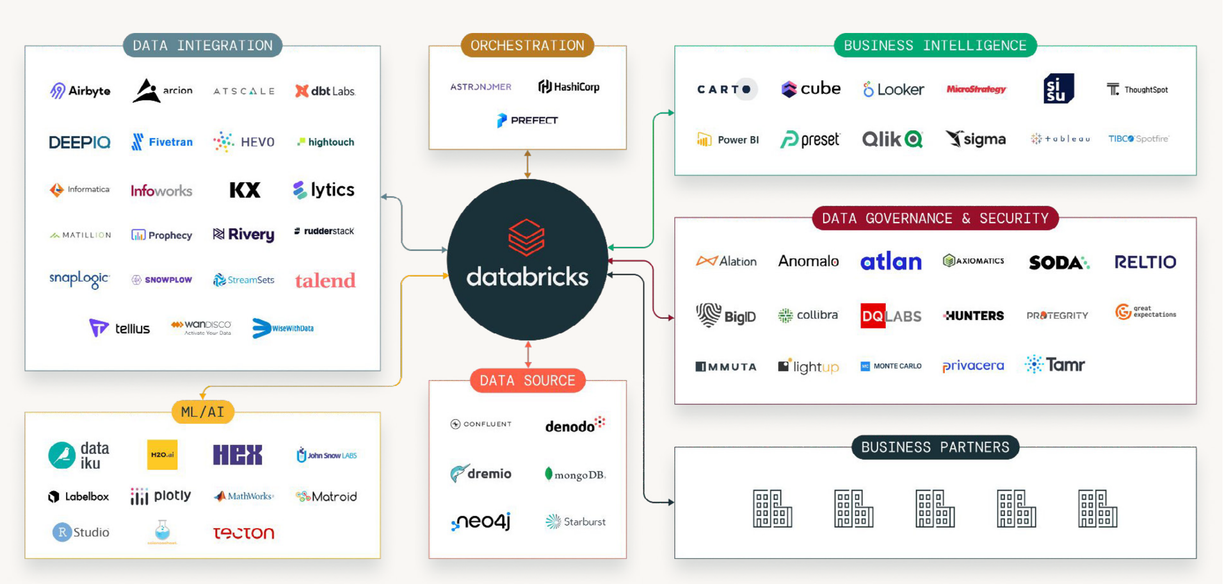


Figure 3
The Databricks Lakehouse Platform integrates with a large collection of technologies

## A rich ecosystem of data solutions

The Databricks Lakehouse Platform is built on open source technologies and uses open standards so leading data solutions can be leveraged with anything you build on the lakehouse. A large collection of technology partners make it easy and simple to integrate the technologies you rely on when migrating to Databricks and to know you are not locked into a closed data technology stack.

## Conclusion

As organizations strive to become data-driven, data engineering is a focal point for success. To deliver reliable, trustworthy data, data engineers shouldn't need to spend time manually developing and maintaining an end-to-end ETL lifecycle. Data engineering teams need an efficient, scalable way to simplify ETL development, improve data reliability and manage operations.

As described, the eight key differentiating capabilities simplify the management of the ETL lifecycle by automating and maintaining all data dependencies, leveraging built-in quality controls with monitoring and by providing deep visibility into pipeline operations with automatic recovery. Data engineering teams can now focus on easily and rapidly building reliable end-to-end production-ready data pipelines using only SQL or Python for batch and streaming that deliver high-value data for analytics, data science or machine learning.

## Follow proven best practices

In the next section, we describe best practices for data engineering end-to end use cases drawn from real-world examples. From data ingestion and real-time processing to analytics and machine learning, you'll learn how to translate raw data into actionable data.

As you explore the rest of this guide, you can find data sets and code samples in the various **Databricks Solution Accelerators**, so you can get your hands dirty as you explore all aspects of the data lifecycle on the Databricks Lakehouse Platform.

**Start experimenting with these free Databricks notebooks.**

SECTION

# 02

# Guidance and Best Practices

SECTION 2.1

# Top 5 Databricks Performance Tips

by BRYAN SMITH and ROB SAKER

March 10, 2022

As solutions architects, we work closely with customers every day to help them get the best performance out of their jobs on Databricks — and we often end up giving the same advice. It's not uncommon to have a conversation with a customer and get double, triple, or even more performance with just a few tweaks. So what's the secret? How are we doing this? Here are the top 5 things we see that can make a huge impact on the performance customers get from Databricks.

Here's a TLDR:

- **Use larger clusters.** It may sound obvious, but this is the number one problem we see. It's actually not any more expensive to use a large cluster for a workload than it is to use a smaller one. It's just faster. If there's anything you should take away from this article, it's this. Read section 1. Really.
- **Use Photon**, Databricks' new, super-fast execution engine. Read section 2 to learn more. You won't regret it.
- **Clean out your configurations**. Configurations carried from one Apache Spark™ version to the next can cause massive problems. Clean up! Read section 3 to learn more.
- **Use Delta Caching**. There's a good chance you're not using caching correctly, if at all. See Section 4 to learn more.
- **Be aware of lazy evaluation**. If this doesn't mean anything to you and you're writing Spark code, jump to section 5.
- **Bonus tip! Table design is super important**. We'll go into this in a future blog, but for now, check out the guide on Delta Lake best practices.

## 1. Give your clusters horsepower!

This is the number one mistake customers make. Many customers create tiny clusters of two workers with four cores each, and it takes forever to do anything. The concern is always the same: they don't want to spend too much money on larger clusters. Here's the thing: **it's actually not any more expensive to use a large cluster for a workload than it is to use a smaller one. It's just faster.**

The key is that you're renting the cluster for the length of the workload. So, if you spin up that two worker cluster and it takes an hour, you're paying for those workers for the full hour. However, if you spin up a four worker cluster and it takes only half an hour, the cost is actually the same! And that trend continues as long as there's enough work for the cluster to do.

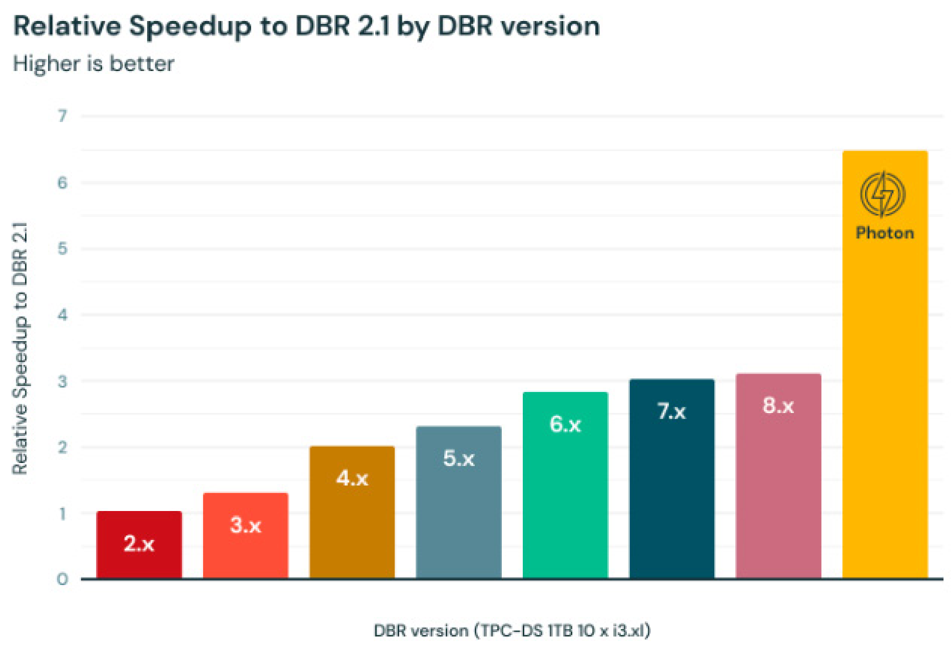
Here's a hypothetical scenario illustrating the point:

| Number of Workers | Cost Per Hour | Length of Workload (hours) | Cost of Workload |
|---|---|---|---|
| 1 | $1 | 2 | $2 |
| 2 | $2 | 1 | $2 |
| 4 | $4 | 0.5 | $2 |
| 8 | $8 | 0.25 | $2 |

Notice that the total cost of the workload stays the same while the real-world time it takes for the job to run drops significantly. So, bump up your Databricks cluster specs and speed up your workloads without spending any more money. It can't really get any simpler than that.

## 2. Use Photon

Our colleagues in engineering have rewritten the Spark execution engine in C++ and dubbed it Photon. The results are impressive!



Beyond the obvious improvements due to running the engine in native code, they've also made use of CPU-level performance features and better memory management. On top of this, they've rewritten the Parquet writer in C++. So this makes writing to Parquet and Delta (based on Parquet) super fast as well!

But let's also be clear about what Photon is speeding up. It improves computation speed for any built-in functions or operations, as well as writes to Parquet or Delta. So joins? Yep! Aggregations? Sure! ETL? Absolutely! That UDF (user-defined function) you wrote? Sorry, but it won't help there. The job that's spending most of its time reading from an ancient on-prem database? Won't help there either, unfortunately.

The good news is that it helps where it can. So even if part of your job can't be sped up, it will speed up the other parts. Also, most jobs are written with the native operations and spend a lot of time writing to Delta, and Photon helps a lot there. So give it a try. You may be amazed by the results!

## 3. Clean out old configurations

You know those Spark configurations you've been carrying along from version to version and no one knows what they do anymore? They may not be harmless. We've seen jobs go from running for hours down to minutes simply by cleaning out old configurations. There may have been a quirk in a particular version of Spark, a performance tweak that has not aged well, or something pulled off some blog somewhere that never really made sense. At the very least, it's worth revisiting your Spark configurations if you're in this situation. Often the default configurations are the best, and they're only getting better. Your configurations may be holding you back.

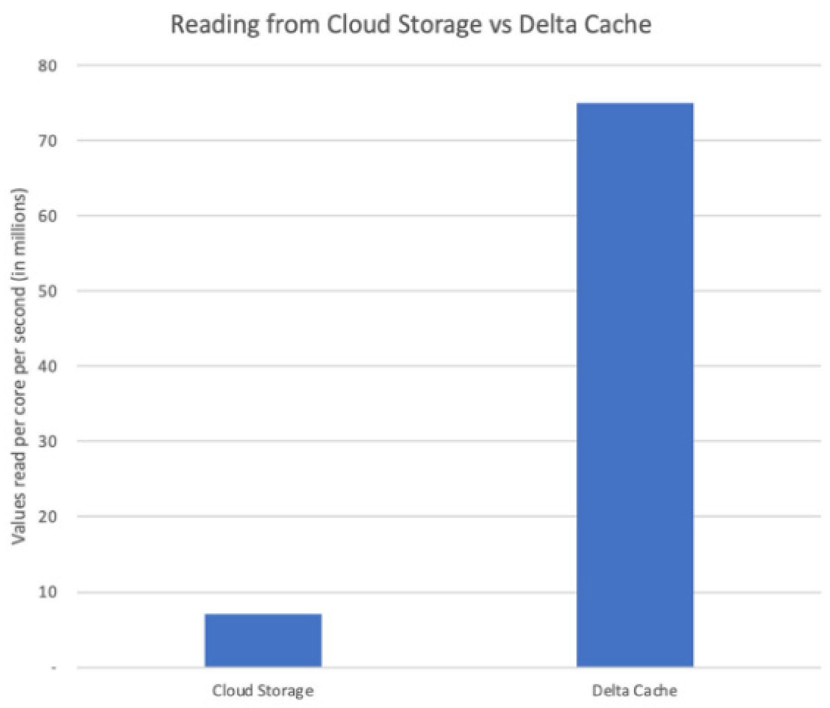
## 4. The Delta Cache is your friend

This may seem obvious, but you'd be surprised how many people are not using the Delta Cache, which loads data off of cloud storage (S3, ADLS) and keeps it on the workers' SSDs for faster access.

If you're using Databricks SQL Endpoints you're in luck. Those have caching on by default. In fact, we recommend using CACHE SELECT * FROM table to preload your "hot" tables when you're starting an endpoint. This will ensure blazing fast speeds for any queries on those tables.

If you're using regular clusters, be sure to use the i3 series on Amazon Web Services (AWS), L series or E series on Azure Databricks, or n2 in GCP. These will all have fast SSDs and caching enabled by default.

Of course, your mileage may vary. If you're doing BI, which involves reading the same tables over and over again, caching gives an amazing boost. However, if you're simply reading a table once and writing out the results as in some ETL jobs, you may not get much benefit. You know your jobs better than anyone. Go forth and conquer.



## 5. Be aware of lazy evaluation

If you're a data analyst or data scientist only using SQL or doing BI you can skip this section. However, if you're in data engineering and writing pipelines or doing processing using Databricks/Spark, read on.

When you're writing Spark code like select, groupBy, filter, etc., you're really building an execution plan. You'll notice the code returns almost immediately when you run these functions. That's because it's not actually doing any computation. So even if you have petabytes of data, it will return in less than a second.

However, once you go to write your results out you'll notice it takes longer. This is due to lazy evaluation. It's not until you try to display or write results that your execution plan is actually run.

```
---------
# Build an execution plan.
# This returns in less than a second but does no work
df2 = (df
  .join(...)
  .select(...)
  .filter(...)
         )

# Now run the execution plan to get results
df2.display()
-------
```

However, there is a catch here. Every time you try to display or write out results, it runs the execution plan again. Let's look at the same block of code but extend it and do a few more operations.

```
---------
# Build an execution plan.
# This returns in less than a second but does no work
df2 = (df
  .join(...)
  .select(...)
  .filter(...)
         )

# Now run the execution plan to get results
df2.display()

# Unfortunately this will run the plan again, including filtering, joining,
etc
df2.display()

# So will this…
df2.count()
-------
```

The developer of this code may very well be thinking that they’re just printing out results three times, but what they’re really doing is kicking off the same processing three times. Oops. That’s a lot of extra work. This is a very common mistake we run into. So why is there lazy evaluation, and what do we do about it?

In short, processing with lazy evaluation is way faster than without it. Databricks/Spark looks at the full execution plan and finds opportunities for optimization that can reduce processing time by orders of magnitude. So that’s great, but how do we avoid the extra computation? The answer is pretty straightforward: save computed results you will reuse.

Let’s look at the same block of code again, but this time let’s avoid the recomputation:

```
# Build an execution plan.
# This returns in less than a second but does no work
df2 = (df
  .join(...)
  .select(...)
  .filter(...)
         )

# save it
df2.write.save(path)

# load it back in
df3 = spark.read.load(path)

# now use it
df3.display()

# this is not doing any extra computation anymore.  No joins, filtering,
etc.  It’s already done and saved.
df3.display()

# nor is this
df3.count()
```

This works especially well when Delta Caching is turned on. In short, you benefit greatly from lazy evaluation, but it’s something a lot of customers trip over. So be aware of its existence and save results you reuse in order to avoid unnecessary computation.

**Start experimenting with these free Databricks notebooks.**

SECTION 2.2

# How to Profile PySpark

by XINRONG MENG, TAKUYA UESHIN, HYUKJIN KWON and ALLAN FOLTING

October 6, 2022

In Apache Spark™, declarative Python APIs are supported for big data workloads. They are powerful enough to handle most common use cases. Furthermore, PySpark UDFs offer more flexibility since they enable users to run arbitrary Python code on top of the Apache Spark™ engine. Users only have to state "what to do"; PySpark, as a sandbox, encapsulates "how to do it." That makes PySpark easier to use, but it can be difficult to identify performance bottlenecks and apply custom optimizations.

To address the difficulty mentioned above, PySpark supports various profiling tools, which are all based on cProfile, one of the standard Python profiler implementations. PySpark Profilers provide information such as the number of function calls, total time spent in the given function, and filename, as well as line number to help navigation. That information is essential to exposing tight loops in your PySpark programs, and allowing you to make performance improvement decisions.

## Driver profiling

PySpark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in the driver program. On the driver side, PySpark is a regular Python process; thus, we can profile it as a normal Python program using cProfile as illustrated below:

```
import cProfile

with cProfile.Profile() as pr:
    # Your code

pr.print_stats()
```

## Workers profiling

Executors are distributed on worker nodes in the cluster, which introduces complexity because we need to aggregate profiles. Furthermore, a Python worker process is spawned per executor for PySpark UDF execution, which makes the profiling more intricate.

The UDF profiler, which is introduced in Spark 3.3, overcomes all those obstacles and becomes a major tool to profile workers for PySpark applications. We'll illustrate how to use the UDF profiler with a simple Pandas UDF example.

Firstly, a PySpark DataFrame with 8,000 rows is generated, as shown below.

```
sdf = spark.range(0, 8 * 1000).withColumn(
  'id', (col('id') % 8).cast('integer') # 1000 rows x 8 groups (if group
by 'id')
).withColumn('v', rand())
```

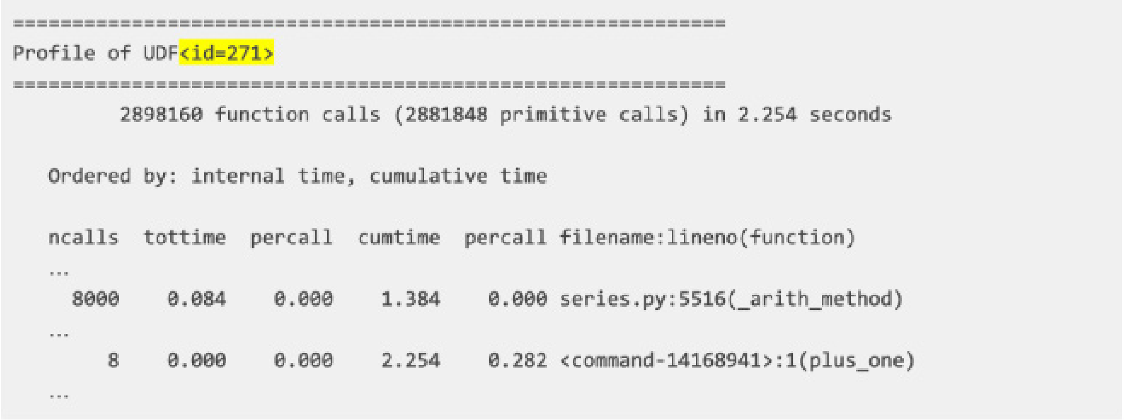
Later, we will group by the id column, which results in 8 groups with 1,000 rows per group.

The Pandas UDF plus_one is then created and applied as shown below:

```
import pandas as pd

def plus_one(pdf: pd.DataFrame) -> pd.DataFrame:
    return pdf.apply(lambda x: x + 1, axis=1)

res = sdf.groupby("id").applyInPandas(plus_one, schema=sdf.schema)
res.collect()
```

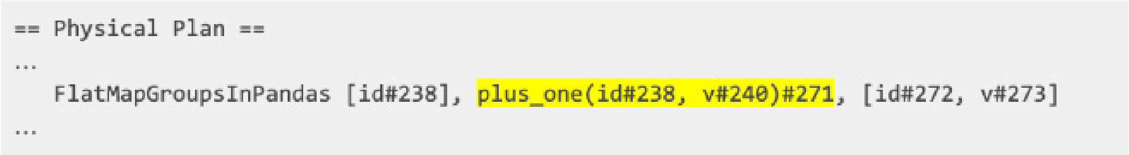
Note that `plus_one` takes a pandas DataFrame and returns another pandas DataFrame. For each group, all columns are passed together as a pandas DataFrame to the `plus_one` UDF, and the returned pandas DataFrames are combined into a PySpark DataFrame.

Executing the example above and running `sc.show_profiles()` prints the following profile. The profile below can also be dumped to disk by `sc.dump_profiles(path)`.



```
===========================================================
Profile of UDF<id=271>
===========================================================
         2898160 function calls (2881848 primitive calls) in 2.254 seconds

   Ordered by: internal time, cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   ...
     8000    0.084    0.000    1.384    0.000 series.py:5516(_arith_method)
   ...
        8    0.000    0.000    2.254    0.282 <command-14168941>:1(plus_one)
   ...
```

The UDF id in the profile (271, highlighted above) matches that in the Spark plan for res. The Spark plan can be shown by calling `res.explain()`.



```
== Physical Plan ==
...
   FlatMapGroupsInPandas [id#238], plus_one(id#238, v#240)#271, [id#272, v#273]
...
```

The first line in the profile's body indicates the total number of calls that were monitored. The column heading includes

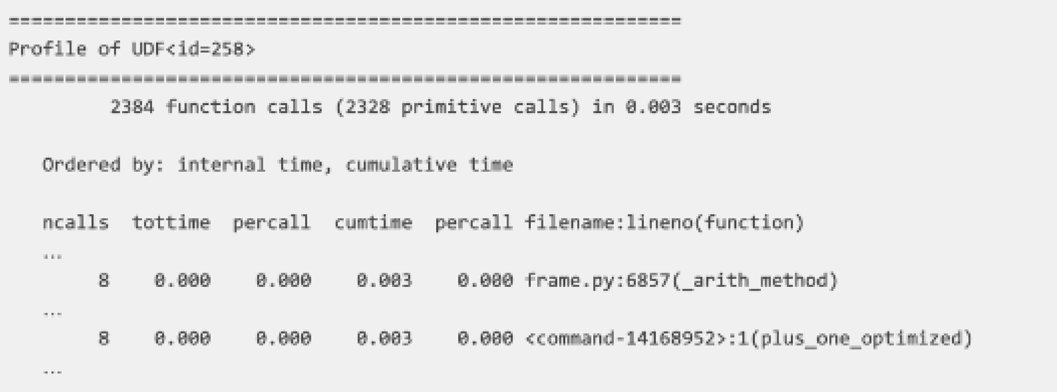
- `ncalls`, for the number of calls.
- `tottime`, for the total time spent in the given function (excluding time spent in calls to sub-functions)
- `percall`, the quotient of tottime divided by ncalls
- `cumtime`, the cumulative time spent in this and all subfunctions (from invocation till exit)
- `percall`, the quotient of cumtime divided by primitive calls
- `filename:lineno(function)`, which provides the respective information for each function

Digging into the column details: `plus_one` is triggered once per group, 8 times in total; `_arith_method` of pandas Series is called once per row, 8,000 times in total. `pandas.DataFrame.apply` applies the function `lambda` x: x + 1 row by row, thus suffering from high invocation overhead.

We can reduce such overhead by substituting the `pandas.DataFrame.apply` with pdf + 1, which is vectorized in pandas. The optimized Pandas UDF looks as follows:

```
import pandas as pd

def plus_one_optimized(pdf: pd.DataFrame) -> pd.DataFrame:
    return pdf + 1

res = sdf.groupby("id").applyInPandas(plus_one_optimized, schema=sdf.
schema)
res.collect()
```

The updated profile is as shown below.



```
============================================================
Profile of UDF<id=258>
============================================================
         2384 function calls (2328 primitive calls) in 0.003 seconds

   Ordered by: internal time, cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   ...
        8    0.000    0.000    0.003    0.000 frame.py:6857(_arith_method)
   ...
        8    0.000    0.000    0.003    0.000 <command-14168952>:1(plus_one_optimized)
   ...
```

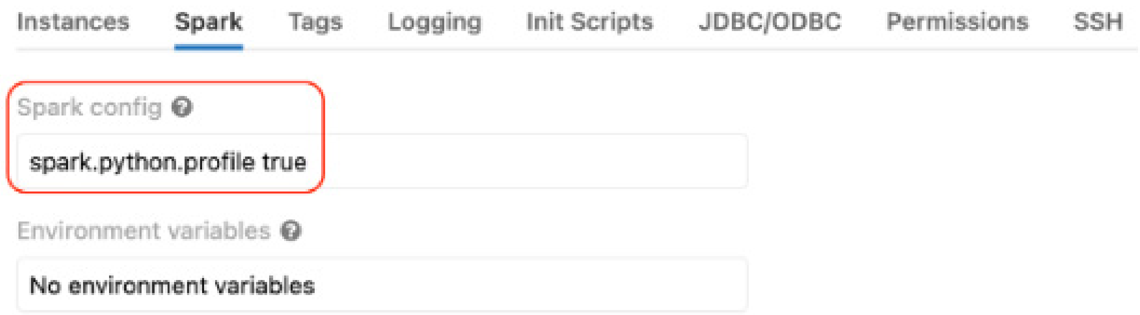
We can summarize the optimizations as follows:

- Arithmetic operation from 8,000 calls to 8 calls
- Total function calls from 2,898,160 calls to 2,384 calls
- Total execution time from 2.300 seconds to 0.004 seconds

The short example above demonstrates how the UDF profiler helps us deeply understand the execution, identify the performance bottleneck and enhance the overall performance of the user-defined function.

The UDF profiler was implemented based on the executor-side profiler, which is designed for PySpark RDD API. The executor-side profiler is available in all active Databricks Runtime versions.

Both the UDF profiler and the executor-side profiler run on Python workers. They are controlled by the `spark.python.profile` Spark configuration, which is false by default. We can enable that Spark configuration on a Databricks Runtime cluster as shown below.



Spark config

spark.python.profile true

Environment variables

No environment variables

## Conclusion

PySpark profilers are implemented based on cProfile; thus, the profile reporting relies on the Stats class. Spark Accumulators also play an important role when collecting profile reports from Python workers.

Powerful profilers are provided by PySpark in order to identify hot loops and suggest potential improvements. They are easy to use and critical to enhance the performance of PySpark programs. The UDF profiler, which is available starting from Databricks Runtime 11.0 (Spark 3.3), overcomes all the technical challenges and brings insights to user-defined functions.

In addition, there is an ongoing effort in the Apache Spark™ open source community to introduce memory profiling on executors; see SPARK-40281 for more information.

**Start experimenting with these free Databricks notebooks.**

SECTION 2.3

# Low-Latency Streaming Data Pipelines With Delta Live Tables and Apache Kafka

by FRANK MUNZ

August 9, 2022

Delta Live Tables (DLT) is the first ETL framework that uses a simple declarative approach for creating reliable data pipelines and fully manages the underlying infrastructure at scale for batch and streaming data. Many use cases require actionable insights derived from near real-time data. Delta Live Tables enables low-latency streaming data pipelines to support such use cases with low latencies by directly ingesting data from event buses like Apache Kafka, AWS Kinesis, Confluent Cloud, Amazon MSK, or Azure Event Hubs.

This article will walk through using DLT with Apache Kafka while providing the required Python code to ingest streams. The recommended system architecture will be explained, and related DLT settings worth considering will be explored along the way.

## Streaming platforms

Event buses or message buses decouple message producers from consumers. A popular streaming use case is the collection of click-through data from users navigating a website where every user interaction is stored as an event in Apache Kafka. The event stream from Kafka is then used for real-time streaming data analytics. Multiple message consumers can read the same data from Kafka and use the data to learn about audience interests, conversion rates, and bounce reasons. The real-time, streaming event data from the user interactions often also needs to be correlated with actual purchases stored in a billing database.

### Apache Kafka

Apache Kafka is a popular open source event bus. Kafka uses the concept of a topic, an append-only distributed log of events where messages are buffered for a certain amount of time. Although messages in Kafka are not deleted once they are consumed, they are also not stored indefinitely. The message retention for Kafka can be configured per topic and defaults to 7 days. Expired messages will be deleted eventually.

This article is centered around Apache Kafka; however, the concepts discussed also apply to many other event busses or messaging systems.

## Streaming data pipelines

In a data flow pipeline, Delta Live Tables and their dependencies can be declared with a standard SQL Create Table As Select (CTAS) statement and the DLT keyword “live.”

When developing DLT with Python, the @dlt.table decorator is used to create a Delta Live Table. To ensure the data quality in a pipeline, DLT uses Expectations which are simple SQL constraints clauses that define the pipeline’s behavior with invalid records.

Since streaming workloads often come with unpredictable data volumes, Databricks employs enhanced autoscaling for data flow pipelines to minimize the overall end-to-end latency while reducing cost by shutting down unnecessary infrastructure.

**Delta Live Tables** are fully recomputed, in the right order, exactly once for each pipeline run.

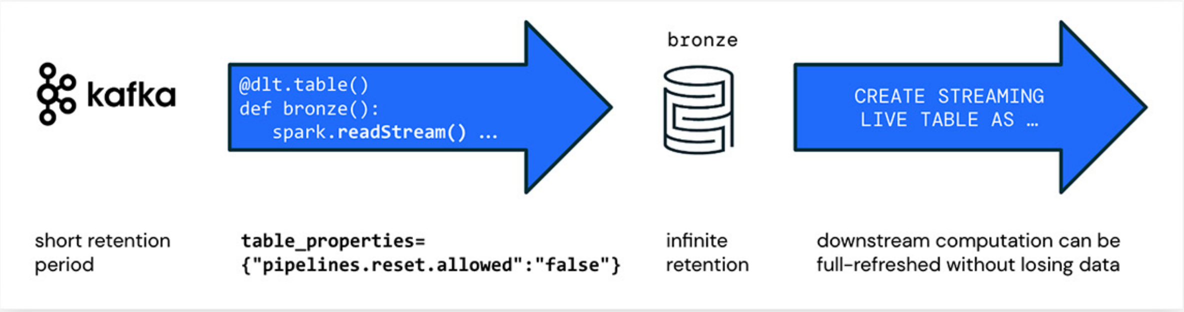
In contrast, **streaming Delta Live Tables** are stateful, incrementally computed and only process data that has been added since the last pipeline run. If the query which defines a streaming live tables changes, new data will be processed based on the new query but existing data is not recomputed. Streaming live tables always use a streaming source and only work over append-only streams, such as Kafka, Kinesis, or Auto Loader. Streaming DLTs are based on top of Spark Structured Streaming.

You can chain multiple streaming pipelines, for example, workloads with very large data volume and low latency requirements.

### Direct ingestion from streaming engines

Delta Live Tables written in Python can directly ingest data from an event bus like Kafka using Spark Structured Streaming. You can set a short retention period for the Kafka topic to avoid compliance issues, reduce costs and then benefit from the cheap, elastic and governable storage that Delta provides.

As a first step in the pipeline, we recommend ingesting the data as is to a Bronze (raw) table and avoid complex transformations that could drop important data. Like any Delta table the Bronze table will retain the history and allow it to perform GDPR and other compliance tasks.


kafka
bronze
short retention period
infinite retention
downstream computation can be full-refreshed without losing data

```
@dlt.table()
def bronze():
   spark.readStream() …
```

```
CREATE STREAMING
LIVE TABLE AS …
```

```
table_properties=
{"pipelines.reset.allowed":"false"}
```

Ingest streaming data from Apache Kafka

When writing DLT pipelines in Python, you use the `@dlt.table` annotation to create a DLT table. There is no special attribute to mark streaming DLTs in Python; simply use `spark.readStream()` to access the stream. Example code for creating a DLT table with the name `kafka_bronze` that is consuming data from a Kafka topic looks as follows:

```python
import dlt
from pyspark.sql.functions import *
from pyspark.sql.types import *

TOPIC = "tracker-events"
KAFKA_BROKER = spark.conf.get("KAFKA_SERVER")
# subscribe to TOPIC at KAFKA_BROKER
raw_kafka_events = (spark.readStream
    .format("kafka")
    .option("subscribe", TOPIC)
    .option("kafka.bootstrap.servers", KAFKA_BROKER)
    .option("startingOffsets", "earliest")
    .load()
    )

@dlt.table(table_properties={"pipelines.reset.allowed":"false"})
def kafka_bronze():
  return raw_kafka_events
```

`pipelines.reset.allowed`

Note that event buses typically expire messages after a certain period of time, whereas Delta is designed for infinite retention.

This might lead to the effect that source data on Kafka has already been deleted when running a full refresh for a DLT pipeline. In this case, not all historic data could be backfilled from the messaging platform, and data would be missing in DLT tables. To prevent dropping data, use the following DLT table property:

`pipelines.reset.allowed=false`

Setting pipelines.reset.allowed to false prevents refreshes to the table but does not prevent incremental writes to the tables or new data from flowing into the table.

## Checkpointing

If you are an experienced Spark Structured Streaming developer, you will notice the absence of checkpointing in the above code. In Spark Structured Streaming checkpointing is required to persist progress information about what data has been successfully processed and upon failure, this metadata is used to restart a failed query exactly where it left off.

Whereas checkpoints are necessary for failure recovery with exactly-once guarantees in Spark Structured Streaming, DLT handles state automatically without any manual configuration or explicit checkpointing required.

## Mixing SQL and Python for a DLT pipeline

A DLT pipeline can consist of multiple notebooks but one DLT notebook is required to be written entirely in either SQL or Python (unlike other Databricks notebooks where you can have cells of different languages in a single notebook).

Now, if your preference is SQL, you can code the data ingestion from Apache Kafka in one notebook in Python and then implement the transformation logic of your data pipelines in another notebook in SQL.

## Schema mapping

When reading data from messaging platform, the data stream is opaque and a schema has to be provided.

The Python example below shows the schema definition of events from a fitness tracker, and how the value part of the Kafka message is mapped to that schema.

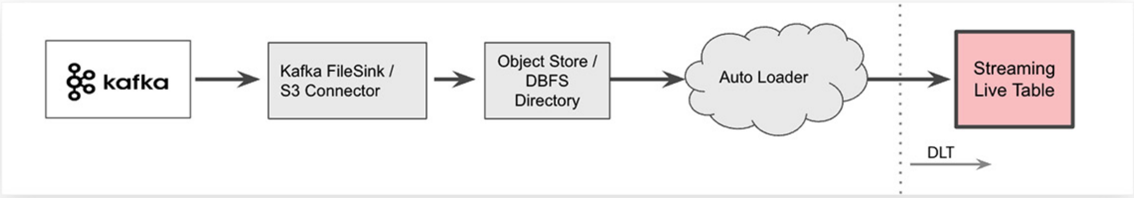
```
event_schema = StructType([ \
    StructField("time", TimestampType(),True)      , \
    StructField("version", StringType(),True), \
    StructField("model", StringType(),True)     , \
    StructField("heart_bpm", IntegerType(),True), \
    StructField("kcal", IntegerType(),True)       \
  ])


# temporary table, visible in pipeline but not in data browser,
# cannot be queried interactively
@dlt.table(comment="real schema for Kakfa payload",
           temporary=True)


def kafka_silver():
  return (
    # kafka streams are (timestamp,value)
    # value contains the kafka payload

    dlt.read_stream("kafka_bronze")
    .select(col("timestamp"),from_json(col("value")
    .cast("string"), event_schema).alias("event"))
    .select("timestamp", "event.*")
  )
```

## Benefits

Reading streaming data in DLT directly from a message broker minimizes the architectural complexity and provides lower end-to-end latency since data is directly streamed from the messaging broker and no intermediary step is involved.

## Streaming ingest with cloud object store intermediary

For some specific use cases, you may want to offload data from Apache Kafka, e.g., using a Kafka connector, and store your streaming data in a cloud object intermediary. In a Databricks workspace, the cloud vendor-specific object-store can then be mapped via the Databricks Files System (DBFS) as a cloud-independent folder. Once the data is offloaded, Databricks Auto Loader can ingest the files.



Auto Loader can ingest data with a single line of SQL code. The syntax to ingest JSON files into a DLT table is shown below (it is wrapped across two lines for readability).

```
-- INGEST with Auto Loader
create or replace streaming live table raw
as select * FROM cloud_files("dbfs:/data/twitter", "json")
```

Note that Auto Loader itself is a streaming data source and all newly arrived files will be processed exactly once, hence the streaming keyword for the raw table that indicates data is ingested incrementally to that table.

Since offloading streaming data to a cloud object store introduces an additional step in your system architecture it will also increase the end-to-end latency and create additional storage costs. Keep in mind that the Kafka connector writing event data to the cloud object store needs to be managed, increasing operational complexity.

Therefore Databricks recommends as a best practice to directly access event bus data from DLT using Spark Structured Streaming as described above.

## Other event buses or messaging systems

This article is centered around Apache Kafka; however, the concepts discussed also apply to other event buses or messaging systems. DLT supports any data source that Databricks Runtime directly supports.

### Amazon Kinesis

In Kinesis, you write messages to a fully managed serverless stream. Same as Kafka, Kinesis does not permanently store messages. The default message retention in Kinesis is one day.

When using Amazon Kinesis, replace format("kafka") with format("kinesis") in the Python code for streaming ingestion above and add Amazon Kinesis-specific settings with option(). For more information, check the section about Kinesis Integration in the Spark Structured Streaming documentation.

### Azure Event Hubs

For Azure Event Hubs settings, check the official documentation at Microsoft and the article Delta Live Tables recipes: Consuming from Azure Event Hubs.

## Summary

DLT is much more than just the "T" in ETL. With DLT, you can easily ingest from streaming and batch sources, cleanse and transform data on the Databricks Lakehouse Platform on any cloud with guaranteed data quality.

Data from Apache Kafka can be ingested by directly connecting to a Kafka broker from a DLT notebook in Python. Data loss can be prevented for a full pipeline refresh even when the source data in the Kafka streaming layer expired.

## Get started

If you are a Databricks customer, simply follow the guide to get started. Read the release notes to learn more about what's included in this GA release. If you are not an existing Databricks customer, sign up for a free trial, and you can view our detailed DLT pricing here.

Join the conversation in the Databricks Community where data-obsessed peers are chatting about Data + AI Summit 2022 announcements and updates. Learn. Network.

Last but not least, enjoy the Dive Deeper into Data Engineering session from the summit. In that session, I walk you through the code of another streaming data example with a Twitter livestream, Auto Loader, Delta Live Tables in SQL, and Hugging Face sentiment analysis.

SECTION 2.4

# Streaming in Production: Collected Best Practices

by BY ANGELA CHU and TRISTEN WENTLING

December 12, 2022

Releasing any data pipeline or application into a production state requires planning, testing, monitoring, and maintenance. Streaming pipelines are no different in this regard; in this blog we present some of the most important considerations for deploying streaming pipelines and applications to a production environment.

At Databricks, we offer two different ways of building and running streaming pipelines and applications — Delta Live Tables (DLT) and Databricks Workflows. DLT is our flagship, fully managed ETL product that supports both batch and streaming pipelines. It offers declarative development, automated operations, data quality, advanced observability capabilities, and more. Workflows enable customers to run Apache Spark™ workloads in Databricks' optimized runtime environment (i.e., Photon) with access to unified governance (Unity Catalog) and storage (Delta Lake). Regarding streaming workloads, both DLT and Workflows share the same core streaming engine — Spark Structured Streaming. In the case of DLT, customers program against the DLT API and DLT uses the Structured Streaming engine under the hood. In the case of Jobs, customers program against the Spark API directly.

The recommendations in this blog post are written from the Structured Streaming engine perspective, most of which apply to both DLT and Workflows (although DLT does take care of some of these automatically, like Triggers and Checkpoints). We group the recommendations under the headings "Before Deployment" and "After Deployment" to highlight when these concepts will need to be applied and are releasing this blog series with this split between the two. There will be additional deep-dive content for some of the sections beyond as well. We recommend reading all sections before beginning work to productionalize a streaming pipeline or application, and revisiting these recommendations as you promote it from dev to QA and eventually production.

## Before deployment

There are many things you need to consider when creating your streaming application to improve the production experience. Some of these topics, like unit testing, checkpoints, triggers, and state management, will determine how your streaming application performs. Others, like naming conventions and how many streams to run on which clusters, have more to do with managing multiple streaming applications in the same environment.

## Unit testing

The cost associated with finding and fixing a bug goes up exponentially the farther along you get in the SDLC process, and a Structured Streaming application is no different. When you're turning that prototype into a hardened production pipeline you need a CI/CD process with built-in tests. So how do you create those tests?

At first you might think that unit testing a streaming pipeline requires something special, but that isn't the case. The general guidance for streaming pipelines is no different than guidance you may have heard for Spark batch jobs. It starts by organizing your code so that it can be unit tested effectively:

- Divide your code into testable chunks
- Organize your business logic into functions calling other functions. If you have a lot of logic in a foreachBatch or you've implemented mapGroupsWithState or flatMapGroupsWithState, organize that code into multiple functions that can be individually tested.
- Do not code in dependencies on the global state or external systems
- Any function manipulating a DataFrame or data set should be organized to take the DataFrame/data set/configuration as input and output the DataFrame/data set

Once your code is separated out in a logical manner you can implement unit tests for each of your functions. Spark-agnostic functions can be tested like any other function in that language. For testing UDFs and functions with DataFrames and data sets, there are multiple Spark testing frameworks available. These frameworks support all of the DataFrame/data set APIs so that you can easily create input, and they have specialized assertions that allow you to compare DataFrame content and schemas. Some examples are:

- The built-in Spark test suite, designed to test all parts of Spark
- spark-testing-base, which has support for both Scala and Python
- spark-fast-tests, for testing Scala Spark 2 & 3
- chispa, a Python version of spark-fast-tests

Code examples for each of these libraries can be found here.

But wait! I'm testing a streaming application here — don't I need to make streaming DataFrames for my unit tests? The answer is no; you do not! Even though a streaming DataFrame represents a data set with no defined ending, when functions are executed on it they are executed on a microbatch — a discrete set of data. You can use the same unit tests that you would use for a batch application, for both stateless and stateful streams. One of the advantages of Structured Streaming over other frameworks is the ability to use the same transformation code for both streaming and with other batch operations for the same sink. This allows you to simplify some operations, like backfilling data, for example, where rather than trying to sync the logic between two different applications, you can just modify the input sources and write to the same destination. If the sink is a Delta table, you can even do these operations concurrently if both processes are append-only operations.

## Triggers

Now that you know your code works, you need to determine how often your stream will look for new data. This is where triggers come in. Setting a trigger is one of the options for the writeStream command, and it looks like this:

```
// Scala/Java
.trigger(Trigger.ProcessingTime("30 seconds"))

# Python
.trigger(processingTime='30 seconds')
```

In the above example, if a microbatch completes in less than 30 seconds, then the engine will wait for the rest of the time before kicking off the next microbatch. If a microbatch takes longer than 30 seconds to complete, then the engine will start the next microbatch immediately after the previous one finishes.

The two factors you should consider when setting your trigger interval are how long you expect your stream to process a microbatch and how often you want the system to check for new data. You can lower the overall processing latency by using a shorter trigger interval and increasing the resources available for the streaming query by adding more workers or using compute or memory optimized instances tailored to your application's performance. These increased resources come with increased costs, so if your goal is to minimize costs, then a longer trigger interval with less compute can work. Normally you would not set a trigger interval longer than what it would typically take for your stream to process a microbatch in order to maximize resource utilization, but setting the interval longer would make sense if your stream is running on a shared cluster and you don't want it to constantly take the cluster resources.

If you do not need your stream to run continuously, either because data doesn't come that often or your SLA is 10 minutes or greater, then you can use the Trigger.Once option. This option will start up the stream, check for anything new since the last time it ran, process it all in one big batch, and then shut down. Just like with a continuously running stream when using Trigger.Once, the checkpoint that guarantees fault tolerance (see below) will guarantee exactly-once processing.

Spark has a new version of Trigger.Once called Trigger.AvailableNow. While Trigger.Once will process everything in one big batch, which depending on your data size may not be ideal, Trigger.AvailableNow will split up the data based on maxFilesPerTrigger and maxBytesPerTrigger settings. This allows the data to be processed in multiple batches. Those settings are ignored with Trigger.Once. You can see examples for setting triggers here.

**Pop quiz —** how do you turn your streaming process into a batch process that automatically keeps track of where it left off with just one line of code?

**Answer —** change your processing time trigger to Trigger.Once/Trigger.AvailableNow! Exact same code, running on a schedule, that will neither miss nor reprocess any records.

## Name your stream

You name your children, you name your pets, now it's time to name your streams. There's a writeStream option called .queryName that allows you to provide a friendly name for your stream. Why bother? Well, suppose you don't name it. In that case, all you'll have to go on in the Structured Streaming tab in the Spark UI is the string <no name> and the unintelligible guid that is automatically generated as the stream's unique identifier. If you have more than one stream running on a cluster, and all of them have <no name> and unintelligible strings as identifiers, how do you find the one you want? If you're exporting metrics how do you tell which is which?

Make it easy on yourself, and name your streams. When you're managing them in production you'll be glad you did, and while you're at it, go and name your batch queries in any foreachBatch() code you have.

## Fault tolerance

How does your stream recover from being shut down? There are a few different cases where this can come into play, like cluster node failures or intentional halts, but the solution is to set up checkpointing. Checkpoints with write-ahead logs provide a degree of protection from your streaming application being interrupted, ensuring it will be able to pick up again where it last left off.

Checkpoints store the current offsets and state values (e.g., aggregate values) for your stream. Checkpoints are stream specific so each should be set to its own location. Doing this will let you recover more gracefully from shutdowns, failures from your application code or unexpected cloud provider failures or limitations.

To configure checkpoints, add the checkpointLocation option to your stream definition:

```
// Scala/Java/Python
streamingDataFrame.writeStream
  .format("delta")
  .option("path", "")
  .queryName("TestStream")
  .option("checkpointLocation", "")
  .start()
```

To keep it simple — every time you call .writeStream, you must specify the checkpoint option with a unique checkpoint location. Even if you're using foreachBatch and the writeStream itself doesn't specify a path or table option, you must still specify that checkpoint. It's how Spark Structured Streaming gives you hassle-free fault tolerance.

Efforts to manage the checkpointing in your stream should be of little concern in general. As Tathagata Das has said, "The simplest way to perform streaming analytics is not having to reason about streaming at all." That said, one setting deserves mention as questions around the maintenance of checkpoint files come up occasionally. Though it is an internal setting that doesn't require direct configuration, the setting spark.sql.streaming.minBatchesToRetain (default 100) controls the number of checkpoint files that get created. Basically, the number of files will be roughly this number times two, as there is a file created noting the offsets at the beginning of the batch (offsets, a.k.a write ahead logs) and another on completing the batch (commits). The number of files is checked periodically for cleanup as part of the internal processes. This simplifies at least one aspect of long-term streaming application maintenance for you.

It is also important to note that some changes to your application code can invalidate the checkpoint. Checking for any of these changes during code reviews before deployment is recommended. You can find examples of changes where this can happen in Recovery Semantics after Changes in a Streaming Query. Suppose you want to look at checkpointing in more detail or consider whether asynchronous checkpointing might improve the latency in your streaming application. In that case, these are covered in greater depth in Speed Up Streaming Queries With Asynchronous State Checkpointing.

### State management and RocksDB

Stateful streaming applications are those where current records may depend on previous events, so Spark has to retain data in between microbatches. The data it retains is called state, and Spark will store it in a state store and read, update and delete it during each microbatch. Typical stateful operations are streaming aggregations, streaming dropDuplicates, stream-stream joins, mapGroupsWithState, or flatMapGroupsWithState. Some common types of examples where you'll need to think about your application state could be sessionization or hourly aggregation using group by methods to calculate business metrics. Each record in the state store is identified by a key that is used as part of the stateful computation, and the more unique keys that are required the larger the amount of state data that will be stored.

When the amount of state data needed to enable these stateful operations grows large and complex, it can degrade your workloads' performance, leading to increased latency or even failures. A typical indicator of the state store being the culprit of added latency is large amounts of time spent in garbage collection (GC) pauses in the JVM. If you are monitoring the microbatch processing time, this could look like a continual increase or wildly varying processing time across microbatches.

The default configuration for a state store, which is sufficient for most general streaming workloads, is to store the state data in the executors' JVM memory. Large number of keys (typically millions, see the Monitoring & Instrumentation section in part 2 of this blog) can add excessive memory pressure on the machine memory and increase the frequency of hitting these GC pauses as it tries to free up resources.

On the Databricks Runtime (now also supported in Apache Spark 3.2+) you can use RocksDB as an alternative state store provider to alleviate this source of memory pressure. RocksDB is an embeddable persistent key-value store for fast storage. It features high performance through a log-structured database engine written entirely in C++ and optimized for fast, low-latency storage.

Leveraging RocksDB as the state store provider still uses machine memory but no longer occupies space in the JVM and makes for a more efficient state management system for large amounts of keys. This doesn't come for free, however, as it introduces an extra step in processing every microbatch. Introducing RocksDB shouldn't be expected to reduce latency except when it is related to memory pressure from state data storage in the JVM. The RocksDB-backed state store still provides the same degree of fault tolerance as the regular state storage as it is included in the stream checkpointing.

RocksDB configuration, like checkpoint configuration, is minimal by design and so you only need to declare it in your overall Spark configuration:

```
spark.conf.set(
  "spark.sql.streaming.stateStore.providerClass",
  "com.databricks.sql.streaming.state.RocksDBStateStoreProvider")
```

If you are monitoring your stream using the streamingQueryListener class, then you will also notice that RocksDB metrics will be included in the stateOperators field. For more detailed information on this see the RocksDB State Store Metrics section of "Structured Streaming in Production."

It's worth noting that large numbers of keys can have other adverse impacts in addition to raising memory consumption, especially with unbounded or non-expiring state keys. With or without RocksDB, the state from the application also gets backed up in checkpoints for fault tolerance. So it makes sense that if you have state files being created so that they will not expire, you will keep accumulating files in the checkpoint, increasing the amount of storage required and potentially the time to write it or recover from failures as well. For the data in memory (see the Monitoring & Instrumentation section in part 2 of this blog) this situation can lead to somewhat vague out-of-memory errors, and for the checkpointed data written to cloud storage you might observe unexpected and unreasonable growth. Unless you have a business need to retain streaming state for all the data that has been processed (and that is rare), read the Spark Structured Streaming documentation and make sure to implement your stateful operations so that the system can drop state records that are no longer needed (pay close attention to dropDuplicates and stream-stream joins).

## Running multiple streams on a cluster

Once your streams are fully tested and configured, it's time to figure out how to organize them in production. It's a common pattern to stack multiple streams on the same Spark cluster to maximize resource utilization and save cost. This is fine to a point, but there are limits to how much you can add to one cluster before performance is affected. The driver has to manage all of the streams running on the cluster, and all streams will compete for the same cores across the workers. You need to understand what your streams are doing and plan your capacity appropriately to stack effectively.

Here is what you should take into account when you're planning on stacking multiple streams on the same cluster:

- Make sure your driver is big enough to manage all of your streams. Is your driver struggling with a high CPU utilization and garbage collection? That means it's struggling to manage all of your streams. Either reduce the number of streams or increase the size of your driver.
- Consider the amount of data each stream is processing. The more data you are ingesting and writing to a sink, the more cores you will need in order to maximize your throughput for each stream. You'll need to reduce the number of streams or increase the number of workers depending on how much data is being processed. For sources like Kafka you will need to configure how many cores are being used to ingest with the minPartitions option if you don't have enough cores for all of the partitions across all of your streams.

- Consider the complexity and data volume of your streams. If all of the streams are doing minimal manipulation and just appending to a sink, then each stream will need fewer resources per microbatch and you'll be able to stack more. If the streams are doing stateful processing or computation/memory-intensive operations, that will require more resources for good performance and you'll want to stack fewer streams.
- Consider scheduler pools. When stacking streams they will all be contending for the same workers and cores, and one stream that needs a lot of cores will cause the other streams to wait. Scheduler pools enable you to have different streams execute on different parts of the cluster. This will enable streams to execute in parallel with a subset of the available resources.
- Consider your SLA. If you have mission critical streams, isolate them as a best practice so lower-criticality streams do not affect them.

On Databricks we typically see customers stack between 10-30 streams on a cluster, but this varies depending on the use case. Consider the factors above so that you can have a good experience with performance, cost and maintainability.

## Conclusion

Some of the ideas we've addressed here certainly deserve their own time and special treatment with a more in-depth discussion, which you can look forward to in later deep dives. However, we hope these recommendations are useful as you begin your journey or seek to enhance your production streaming experience. Be sure to continue with the next post, "Streaming in Production: Collected Best Practices, Part 2."

**Review Databrick's Structured Streaming Getting Started Guide** ▶

**Start experimenting with these free Databricks notebooks.**

SECTION 2.5

# Streaming in Production: Collected Best Practices, Part 2

by ANGELA CHU and TRISTEN WENTLING

January 10, 2023

In our two-part blog series titled "Streaming in Production: Collected Best Practices," this is the second article. Here we discuss the "After Deployment" considerations for a Structured Streaming Pipeline. The majority of the suggestions in this post are relevant to both Structured Streaming Jobs and Delta Live Tables (our flagship and fully managed ETL product that supports both batch and streaming pipelines).

## After deployment

After the deployment of your streaming application, there are typically three main things you'll want to know:

- How is my application running?
- Are resources being used efficiently?
- How do I manage any problems that come up?

We'll start with an introduction to these topics, followed by a deeper dive later in this blog series.

### Monitoring and instrumentation (How is my application running?)

Streaming workloads should be pretty much hands-off once deployed to production. However, one thing that may sometimes come to mind is: "how is my application running?" Monitoring applications can take on different levels and forms depending on:

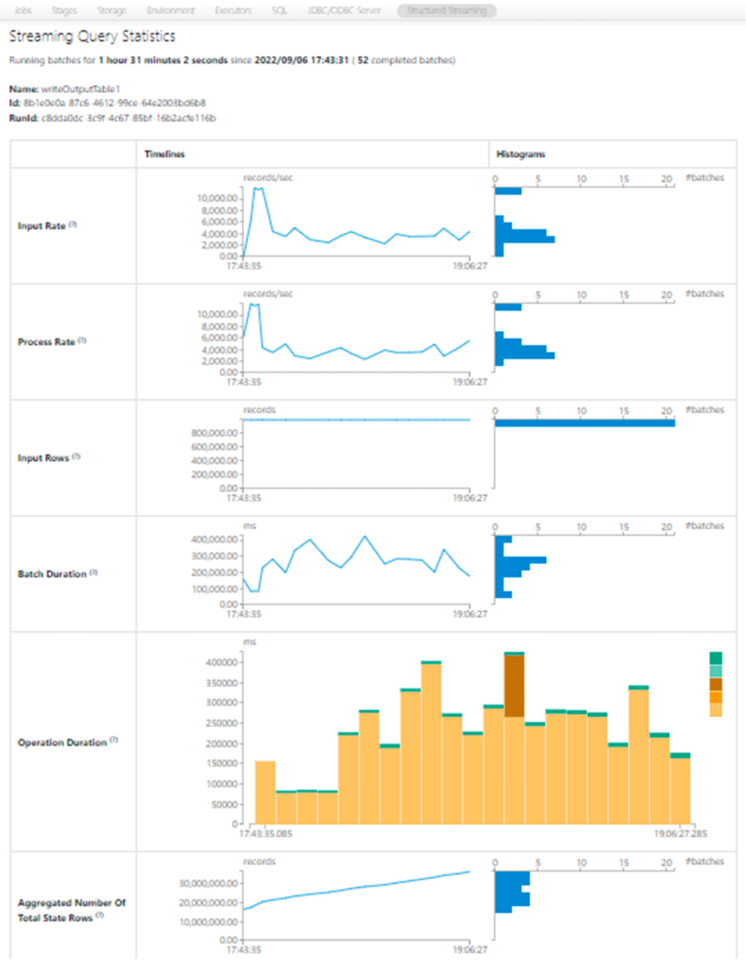
- the metrics collected for your application (batch duration/latency, throughput, …)
- where you want to monitor the application from

At the simplest level, there is a streaming dashboard (A Look at the New Structured Streaming UI) and built-in logging directly in the Spark UI that can be used in a variety of situations.

This is in addition to setting up failure alerts on jobs running streaming workloads.

If you want more fine-grained metrics or to create custom actions based on these metrics as part of your code base, then the `StreamingQueryListener` is better aligned with what you're looking for.

If you want the Spark metrics to be reported (including machine level traces for drivers or workers) you should use the platform's metrics sink.



Streaming Query Statistics

Running batches for **1 hour 31 minutes 2 seconds** since **2022/09/06 17:43:31** ( **52** completed batches)

**Name:** writeOutputTable1
**Id:** 8b1e0e0a-87c6-4612-99ce-64e2003bd6b8
**RunId:** c8dda0dc-3c9f-4c67-85bf-16b2acfe116b

The Apache Spark Structured Streaming UI

Another point to consider is where you want to surface these metrics for observability. There is a Ganglia dashboard at the cluster level, integrated partner applications like Datadog for monitoring streaming workloads, or even more open source options you can build using tools like Prometheus and Grafana. Each has advantages and disadvantages to consider around cost, performance, and maintenance requirements.

Whether you have low volumes of streaming workloads where interactions in the UI are sufficient or have decided to invest in a more robust monitoring platform, you should know how to observe your production streaming workloads. Further "Monitoring and Alerting" posts later in this series will contain a more thorough discussion. In particular, we'll see different measures on which to monitor streaming applications and then later take a deeper look at some of the tools you can leverage for observability.

## Application optimization (Are resources being used effectively? Think "cost")

The next concern we have after deploying to production is "is my application using resources effectively?" As developers, we understand (or quickly learn) the distinction between working code and well-written code. Improving the way your code runs is usually very satisfying, but what ultimately matters is the overall cost of running it. Cost considerations for Structured Streaming applications will be largely similar to those for other Spark applications. One notable difference is that failing to optimize for production workloads can be extremely costly, as these workloads are frequently "always-on" applications, and thus wasted expenditure can quickly compound. Because assistance with cost optimization is frequently requested, a separate post in this series will address it. The key points that we'll focus on will be efficiency of usage and sizing.

Getting the cluster sizing right is one of the most significant differences between efficiency and wastefulness in streaming applications. This can be particularly tricky because in some cases it's difficult to estimate the full load conditions of the application in production before it's actually there. In other cases, it may be difficult due to natural variations in volume handled throughout the day, week, or year. When first deploying, it can be beneficial to oversize slightly, incurring the extra expense to avoid inducing performance bottlenecks. Utilize the monitoring tools you chose to employ after the cluster has been running for a few weeks to ensure proper cluster utilization. For example, are CPU and memory levels being used at a high level during peak load or is the load generally small and the cluster may be downsized? Maintain regular monitoring of this and keep an eye out for changes in data volume over time; if either occurs, a cluster resize may be required to maintain cost-effective operation.

As a general guideline, you should avoid excessive shuffle operations, joins, or an excessive or extreme watermark threshold (don't exceed your needs), as each can increase the number of resources you need to run your application. A large watermark threshold will cause Structured Streaming to keep more data in the state store between batches, leading to an increase in memory requirements across the cluster. Also, pay attention to the type of VM configured — are you using memory-optimized for your memory-intense stream? Compute-optimized for your computationally-intensive stream? If not, look at the utilization levels for each and consider trying a machine type that could be a better fit. Newer families of servers from cloud providers with more optimal CPUs often lead to faster execution, meaning you might need fewer of them to meet your SLA.

## Troubleshooting (How do I manage any problems that come up?)

The last question we ask ourselves after deployment is "how do I manage any problems that come up?" As with cost optimization, troubleshooting streaming applications in Spark often looks the same as other applications since most of the mechanics remain the same under the hood. For streaming applications, issues usually fall into two categories — failure scenarios and latency scenarios

### Failure scenarios

Failure scenarios typically manifest with the stream stopping with an error, executors failing or a driver failure causing the whole cluster to fail. Common causes for this are:

- Too many streams running on the same cluster, causing the driver to be overwhelmed. On Databricks, this can be seen in Ganglia, where the driver node will show up as overloaded before the cluster fails.
- Too few workers in a cluster or a worker size with too small of a core-to-memory ratio, causing executors to fail with an Out Of Memory error. This can also be seen on Databricks in Ganglia before an executor fails, or in the Spark UI under the executors tab.
- Using a collect to send too much data to the driver, causing it to fail with an Out Of Memory error.

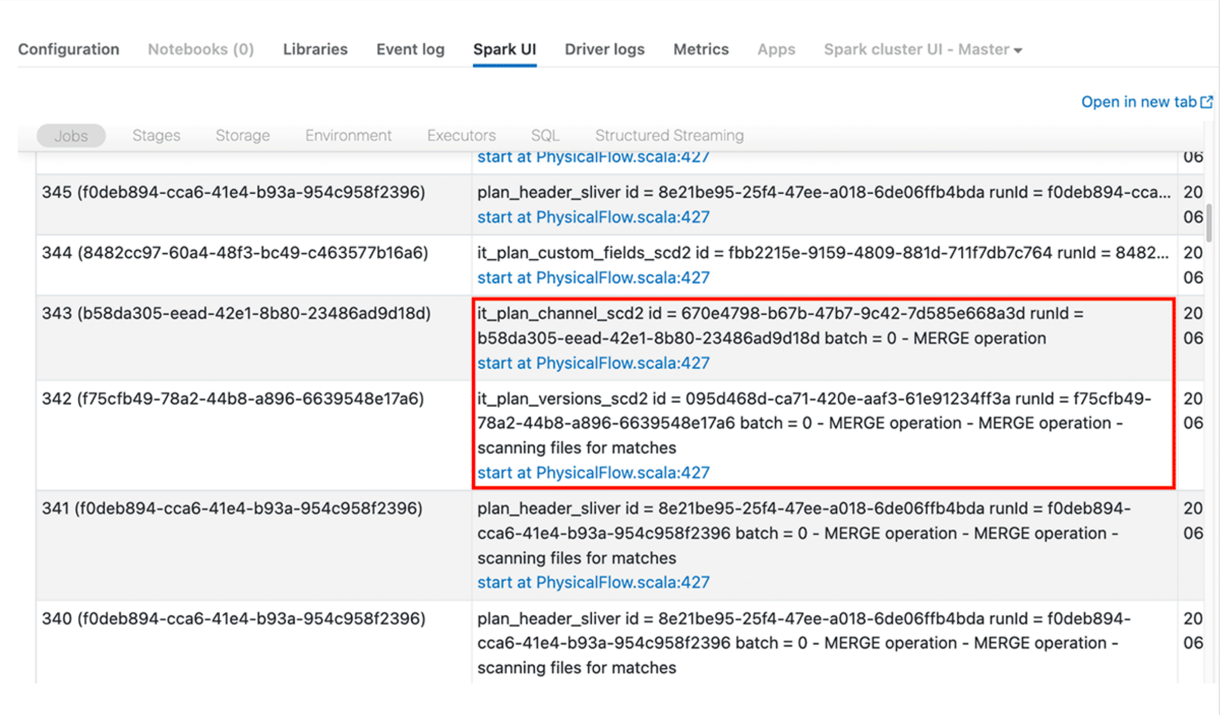
## Latency scenarios

For latency scenarios, your stream will not execute as fast as you want or expect. A latency issue can be intermittent or constant. Too many streams or too small of a cluster can be the cause of this as well. Some other common causes are:

- Data skew — when a few tasks end up with much more data than the rest of the tasks. With skewed data, these tasks take longer to execute than the others, often spilling to disk. Your stream can only run as fast as its slowest task.
- Executing a stateful query without defining a watermark or defining a very long one will cause your state to grow very large, slowing down your stream over time and potentially leading to failure.
- Poorly optimized sink. For example, performing a merge into an over-partitioned Delta table as part of your stream.
- Stable but high latency (batch execution time). Depending on the cause, adding more workers to increase the number of cores concurrently available for Spark tasks can help. Increasing the number of input partitions and/or decreasing the load per core through batch size settings can also reduce the latency.

Just like troubleshooting a batch job, you'll use Ganglia to check cluster utilization and the Spark UI to find performance bottlenecks. There is a specific Structured Streaming tab in the Spark UI created to help monitor and troubleshoot streaming applications. On that tab each stream that is running will be listed, and you'll see either your stream name if you named your stream or <no name> if you didn't. You'll also see a stream ID that will be visible on the Jobs tab of the Spark UI so that you can tell which jobs are for a given stream.

You'll notice above we said which jobs are for a given stream. It's a common misconception that if you were to look at a streaming application in the Spark UI you would just see one job in the Jobs tab running continuously. Instead, depending on your code, you will see one or more jobs that start and complete for each microbatch. Each job will have the stream ID from the Structured Streaming tab and a microbatch number in the description, so you'll be able to tell which jobs go with which stream. You can click into those jobs to find the longest running stages and tasks, check for disk spills, and search by Job ID in the SQL tab to find the slowest queries and check their explain plans.



| | | |
|---|---|---|
| | start at PhysicalFlow.scala:427 | 06 |
| 345 (f0deb894-cca6-41e4-b93a-954c958f2396) | plan_header_sliver id = 8e21be95-25f4-47ee-a018-6de06ffb4bda runId = f0deb894-cca...<br>start at PhysicalFlow.scala:427 | 20<br>06 |
| 344 (8482cc97-60a4-48f3-bc49-c463577b16a6) | it_plan_custom_fields_scd2 id = fbb2215e-9159-4809-881d-711f7db7c764 runId = 8482...<br>start at PhysicalFlow.scala:427 | 20<br>06 |
| 343 (b58da305-eead-42e1-8b80-23486ad9d18d) | it_plan_channel_scd2 id = 670e4798-b67b-47b7-9c42-7d585e668a3d runId = b58da305-eead-42e1-8b80-23486ad9d18d batch = 0 - MERGE operation<br>start at PhysicalFlow.scala:427 | 20<br>06 |
| 342 (f75cfb49-78a2-44b8-a896-6639548e17a6) | it_plan_versions_scd2 id = 095d468d-ca71-420e-aaf3-61e91234ff3a runId = f75cfb49-78a2-44b8-a896-6639548e17a6 batch = 0 - MERGE operation - MERGE operation - scanning files for matches<br>start at PhysicalFlow.scala:427 | 20<br>06 |
| 341 (f0deb894-cca6-41e4-b93a-954c958f2396) | plan_header_sliver id = 8e21be95-25f4-47ee-a018-6de06ffb4bda runId = f0deb894-cca6-41e4-b93a-954c958f2396 batch = 0 - MERGE operation - MERGE operation - scanning files for matches<br>start at PhysicalFlow.scala:427 | 20<br>06 |
| 340 (f0deb894-cca6-41e4-b93a-954c958f2396) | plan_header_sliver id = 8e21be95-25f4-47ee-a018-6de06ffb4bda runId = f0deb894-cca6-41e4-b93a-954c958f2396 batch = 0 - MERGE operation - MERGE operation - scanning files for matches | 20<br>06 |

The Jobs tab in the Apache Spark UI

If you click on your stream in the Structured Streaming tab you'll see how much time the different streaming operations are taking for each microbatch, such as adding a batch, query planning and committing (see earlier screenshot of the Apache Spark Structured Streaming UI). You can also see how many rows are being processed as well as the size of your state store for a stateful stream. This can give insights into where potential latency issues are.

We will go more in-depth with troubleshooting later in this blog series, where we'll look at some of the causes and remedies for both failure scenarios and latency scenarios as we outlined above.

## Conclusion

You may have noticed that many of the topics covered here are very similar to how other production Spark applications should be deployed. Whether your workloads are primarily streaming applications or batch processes, the majority of the same principles will apply. We focused more on things that become especially important when building out streaming applications, but as we're sure you've noticed by now, the topics we discussed should be included in most production deployments.

Across the majority of industries in the world today information is needed faster than ever, but that won't be a problem for you. With Spark Structured Streaming you're set to make it happen at scale in production. Be on the lookout for more in-depth discussions on some of the topics we've covered in this blog, and in the meantime keep streaming!

**Review Databricks Structured Streaming in Production Documentation**

**Start experimenting with these free Databricks notebooks.**

SECTION 2.6

# Building Geospatial Data Products

by MILOS COLIC

January 6, 2023

Geospatial data has been driving innovation for centuries, through use of maps, cartography and more recently through digital content. For example, the oldest map has been found etched in a piece of mammoth tusk and dates approximately 25,000 BC. This makes geospatial data one of the oldest data sources used by society to make decisions. A more recent example, labeled as the birth of spatial analysis, is that of Charles Picquet in 1832 who used geospatial data to analyze Cholera outbreaks in Paris; a couple of decades later John Snow in 1854 followed the same approach for Cholera outbreaks in London. These two individuals used geospatial data to solve one of the toughest problems of their times and in effect save countless lives. Fast-forwarding to the 20th century, the concept of Geographic Information Systems (GIS) was first introduced in 1967 in Ottawa, Canada, by the Department of Forestry and Rural Development.

Today we are in the midst of the cloud computing industry revolution — supercomputing scale available to any organization, virtually infinitely scalable for both storage and compute. Concepts like data mesh and data marketplace are emerging within the data community to address questions like platform federation and interoperability. How can we adopt these concepts to geospatial data, spatial analysis and GIS systems? By adopting the concept of data products and approaching the design of geospatial data as a product.

In this blog we will provide a point of view on how to design scalable geospatial data products that are modern and robust. We will discuss how Databricks Lakehouse Platform can be used to unlock the full potential of geospatial products that are one of the most valuable assets in solving the toughest problems of today and the future.

## What is a data product? And how to design one?

The most broad and the most concise definition of a "data product" was coined by DJ Patil (the first U.S. Chief Data Scientist) in *Data Jujitsu: The Art of Turning Data into Product:* "a product that facilitates an end goal through the use of data." The complexity of this definition (as admitted by Patil himself) is needed to encapsulate the breadth of possible products, to include dashboards, reports, Excel spreadsheets, and even CSV extracts shared via emails. You might notice that the examples provided deteriorate rapidly in quality, robustness and governance.

What are the concepts that differentiate a successful product versus an unsuccessful one? Is it the packaging? Is it the content? Is it the quality of the content? Or is it only the product adoption in the market? Forbes defines the 10 must-haves of a successful product. A good framework to summarize this is through the value pyramid.

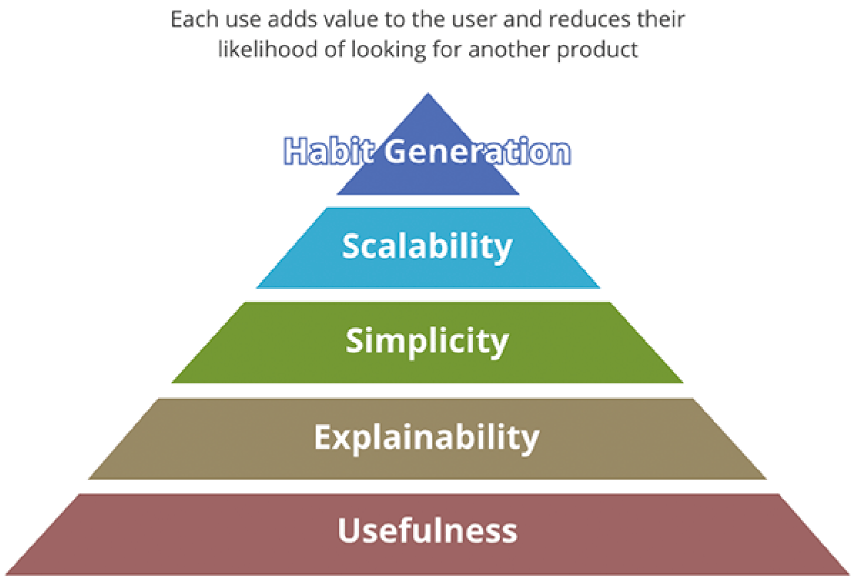

Each use adds value to the user and reduces their likelihood of looking for another product
Habit Generation
Scalability
Simplicity
Explainability
Usefulness

Figure 1: Product value pyramid (source)

The value pyramid provides a priority on each aspect of the product. Not every value question we ask about the product carries the same amount of weight. If the output is not useful none of the other aspects matter — the output isn't really a product but becomes more of a data pollutant to the pool of useful results. Likewise, scalability only matters after simplicity and explainability are addressed.

How does the value pyramid relate to the data products? Each data output, in order to be a data product:

- **Should have clear usefulness.** The amount of the data society is generating is rivaled only by the amount of data pollutants we are generating. These are outputs lacking clear value and use, much less a strategy for what to do with them.
- **Should be explainable.** With the emergence of AI/ML, explainability has become even more important for data driven decision-making. Data is as good as the metadata describing it. Think of it in terms of food — taste does matter, but a more important factor is the nutritional value of ingredients.
- **Should be simple.** An example of product misuse is using a fork to eat cereal instead of using a spoon. Furthermore, simplicity is essential but not sufficient — beyond simplicity the products should be intuitive. Whenever possible both intended and unintended uses of the data should be obvious.
- **Should be scalable.** Data is one of the few resources that grows with use. The more data you process the more data you have. If both inputs and outputs of the system are unbounded and ever-growing, then the system has to be scalable in compute power, storage capacity and compute expressive power. Cloud data platforms like Databricks are in a unique position to answer for all of the three aspects.
- **Should generate habits.** In the data domain we are not concerned with customer retention as is the case for the retail products. However, the value of habit generation is obvious if applied to best practices. The systems and data outputs should exhibit the best practices and promote them — it should be easier to use the data and the system in the intended way than the opposite.

The geospatial data should adhere to all the aforementioned aspects — any data products should. On top of this tall order, geospatial data has some specific needs.

## Geospatial data standards

Geospatial data standards are used to ensure that geographic data is collected, organized, and shared in a consistent and reliable way. These standards can include guidelines for things like data formatting, coordinate systems, map projections, and metadata. Adhering to standards makes it easier to share data between different organizations, allowing for greater collaboration and broader access to geographic information.

The Geospatial Commision (UK government) has defined the UK Geospatial Data Standards Register as a central repository for data standards to be applied in the case of geospatial data. Furthermore, the mission of this registry is to:

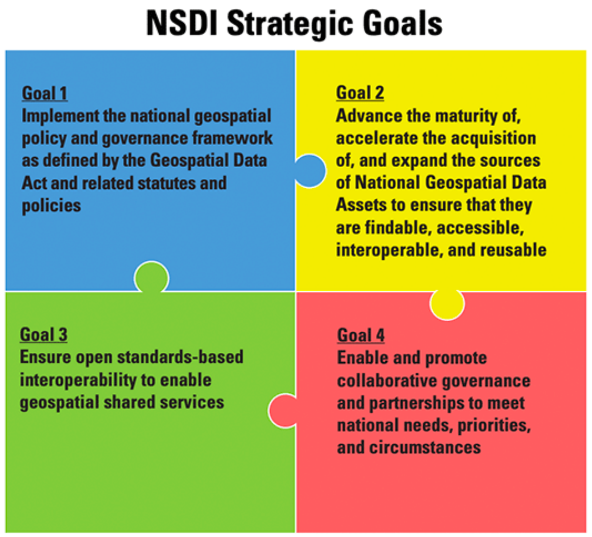
- **"Ensure UK geospatial data is more consistent and coherent and usable across a wider range of systems."** — These concepts are a callout for the importance of explainability, usefulness and habit generation (possibly other aspects of the value pyramid).
- **"Empower the UK geospatial community to become more engaged with the relevant standards and standards bodies."** — Habit generation within the community is as important as the robust and critical design on the standard. If not adopted standards are useless.
- **"Advocate the understanding and use of geospatial data standards within other sectors of government."** — Value pyramid applies to the standards as well — concepts like ease of adherence (usefulness/simplicity), purpose of the standard (explainability/usefulness), adoption (habit generation) are critical for the value generation of a standard.

A critical tool for achieving the data standards mission is the FAIR data principles:

- **Findable** — The first step in (re)using data is to find them. Metadata and data should be easy to find for both humans and computers. Machine-readable metadata are essential for automatic discovery of data sets and services.
- **Accessible** — Once the user finds the required data, she/he/they need to know how they can be accessed, possibly including authentication and authorization.
- **Interoperable** — The data usually needs to be integrated with other data. In addition, the data needs to interoperate with applications or workflows for analysis, storage, and processing.
- **Reusable** — The ultimate goal of FAIR is to optimize the reuse of data. To achieve this, metadata and data should be well-described so that they can be replicated and/or combined in different settings.

We share the belief that the FAIR principles are crucial for the design of scalable data products we can trust. To be fair, FAIR is based on common sense, so why is it key to our considerations? *"What I see in FAIR is not new in itself, but what it does well is to articulate, in an accessible way, the need for a holistic approach to data improvement. This ease in communication is why FAIR is being used increasingly widely as an umbrella for data improvement — and not just in the geospatial community."* — A FAIR wind sets our course for data improvement.

To further support this approach, the Federal Geographic Data Committee has developed the National Spatial Data Infrastructure (NSDI) Strategic Plan that covers the years 2021-2024 and was approved in November 2020. The goals of NSDI are in essence FAIR principles and convey the same message of designing systems that promote the circular economy of data — data products that flow between organizations following common standards and in each step through the data supply chain unlock new value and new opportunities. The fact that these principles are permeating different jurisdictions and are adopted across different regulators is a testament to the robustness and soundness of the approach.



NSDI Strategic Goals

Goal 1
Implement the national geospatial policy and governance framework as defined by the Geospatial Data Act and related statutes and policies

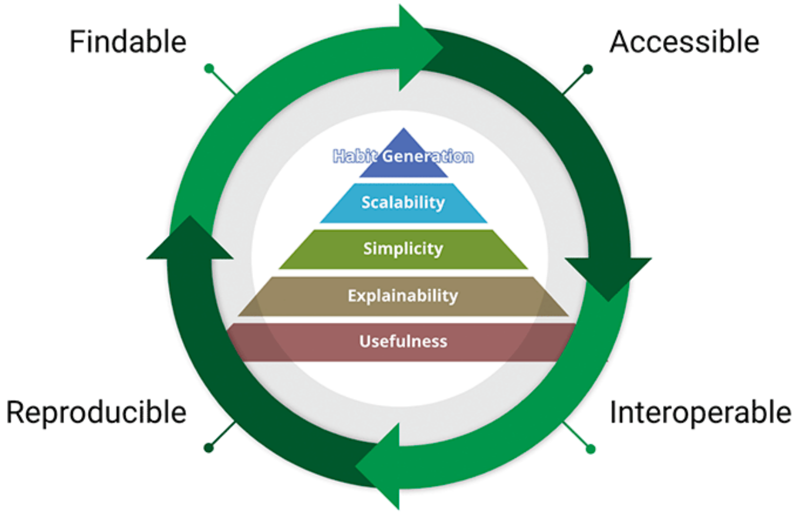
Goal 2
Advance the maturity of, accelerate the acquisition of, and expand the sources of National Geospatial Data Assets to ensure that they are findable, accessible, interoperable, and reusable

Goal 3
Ensure open standards-based interoperability to enable geospatial shared services

Goal 4
Enable and promote collaborative governance and partnerships to meet national needs, priorities, and circumstances

Figure 2:
NDSI Strategic Goals

The FAIR concepts weave really well together with the data product design. In fact FAIR is traversing the whole product value pyramid and forms a value cycle. By adopting both the value pyramid and FAIR principles we design data products with both internal and external outlook. This promotes data reuse as opposed to data accumulation.



Why do FAIR principles matter for geospatial data and geospatial data products? FAIR is transcendent to geospatial data, it is actually transcendent to data, it is a simple yet coherent system of guiding principles for good design — and that good design can be applied to anything including geospatial data and geospatial systems.

## Grid index systems

In traditional GIS solutions' performance of spatial operations are usually achieved by building tree structures (KD trees, ball trees, Quad trees, etc). The issue with tree approaches is that they eventually break the scalability principle — when the data is too big to be processed in order to build the tree and the computation required to build the tree is too long and defeats the purpose. This also negatively affects the accessibility of data; if we cannot construct the tree we cannot access the complete data and in effect we cannot reproduce the results. In this case, grid index systems provide a solution.

Grid index systems are built from the start with the scalability aspects of the geospatial data in mind. Rather than building the trees, they define a series of grids that cover the area of interest. In the case of H3 (pioneered by Uber), the grid covers the area of the Earth; in the case of local grid index systems (e.g., British National Grid) they may only cover the specific area of interest. These grids are composed of cells that have unique identifiers. There is a mathematical relationship between location and the cell in the grid. This makes the grid index systems very scalable and parallel in nature.

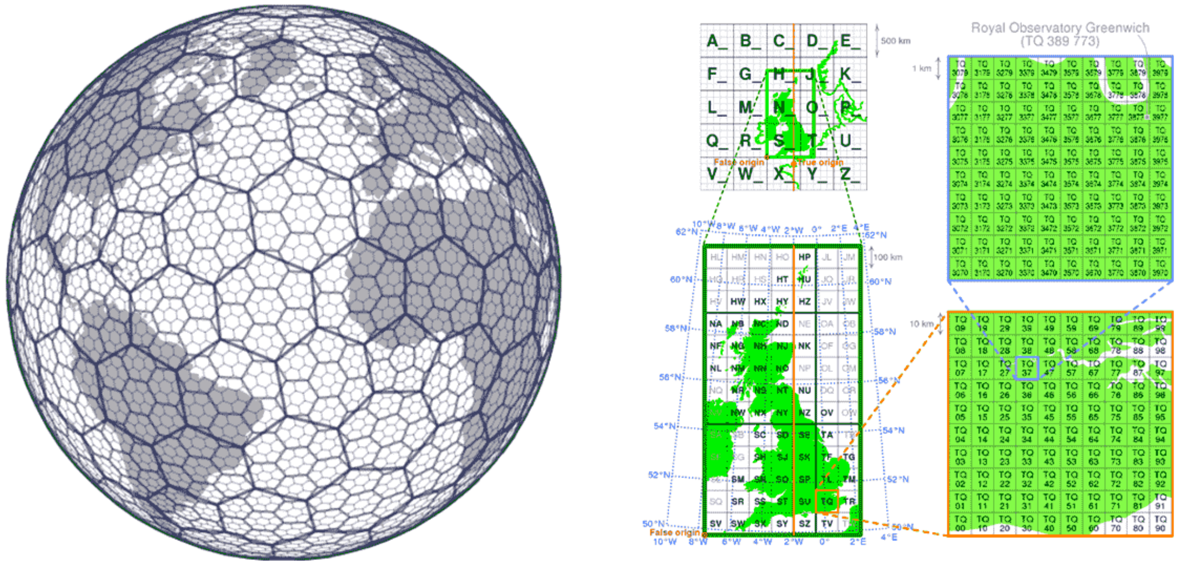


Figure 4: Grid Index Systems (H3, British National Grid)

Another important aspect of grid index systems is that they are open source, allowing index values to be universally leveraged by data producers and consumers alike. Data can be enriched with the grid index information at any step of its journey through the data supply chain. This makes the grid index systems an example of community driven data standards. Community driven data standards by nature do not require enforcement, which fully adheres to the habit generation aspect of value pyramid and meaningfully addresses interoperability and accessibility principles of FAIR.

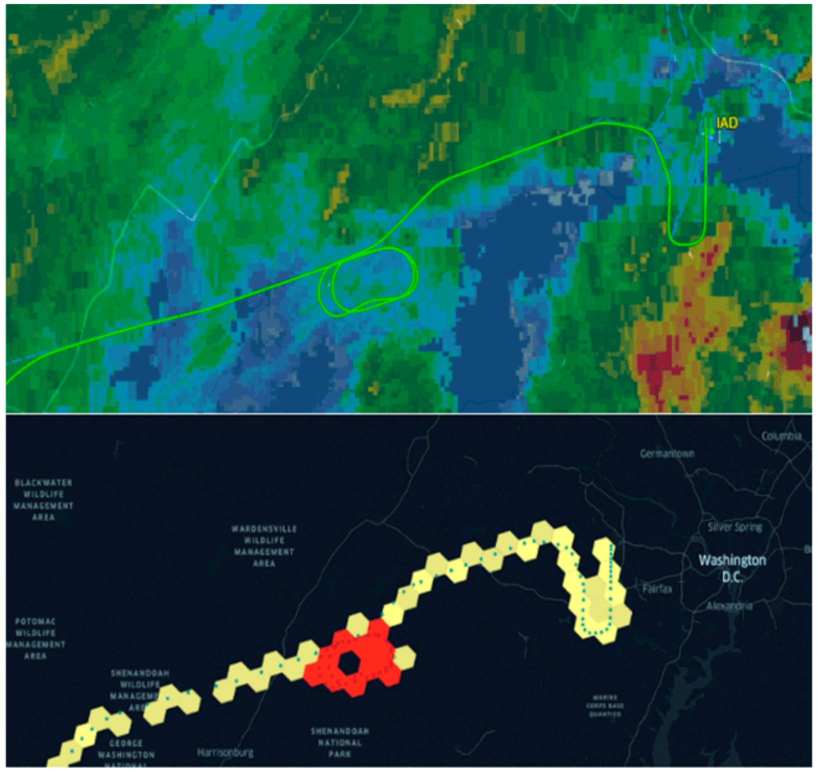

IAD
Columbia
Germantown
BLACKWATER WILDLIFE MANAGEMENT AREA
WARDENSVILLE WILDLIFE MANAGEMENT AREA
Silver Spring
Washington D.C.
Fairfax
Alexandria
POTOMAC WILDLIFE MANAGEMENT AREA
SHENANDOAH WILDLIFE MANAGEM ARE
MARINE CORPS BASE QUANTICO
GEORGE WASHINGTON NATIONAL
Harrisonburg
SHENANDOAH NATIONAL PARK

Figure 5: Example of using H3 to express flight holding patterns

Databricks has recently announced native support for the H3 grid index system following the same value proposition. Adopting common industry standards driven by the community is the only way to properly drive habit generation and interoperability. To strengthen this statement, organizations like CARTO , ESRI and Google have been promoting the usage of grid index systems for scalable GIS system design. In addition, Databricks Labs project Mosaic supports the British National Grid as the standard grid index system that is widely used in the UK government. Grid index systems are key for the scalability of geospatial data processing and for properly designing solutions for complex problems (e.g., figure 5 — flight holding patterns using H3).

## Geospatial data diversity

Geospatial data standards spend a solid amount of effort regarding data format standardization, and format for that matter is one of the most important considerations when it comes to interoperability and reproducibility. Furthermore, if the reading of your data is complex — how can we talk about simplicity? Unfortunately geospatial data formats are typically complex, as data can be produced in a number of formats including both open source and vendor-specific formats. Considering only vector data, we can expect data to arrive in WKT, WKB, GeoJSON, web CSV, CSV, Shape File, GeoPackage, and many others. On the other hand, if we are considering raster data we can expect data to arrive in any number of formats such as GeoTiff, netCDF, GRIB, or GeoDatabase; for a comprehensive list of formats please consult this blog.

Geospatial data domain is so diverse and has organically grown over the years around the use cases it was addressing. Unification of such a diverse ecosystem is a massive challenge. A recent effort by the Open Geospatial Consortium (OGC) to standardize to Apache Parquet and its geospatial schema specification GeoParquet is a step in the right direction. Simplicity is one of the key aspects of designing a good scalable and robust product — unification leads to simplicity and addresses one of the main sources of friction in the ecosystem — the data ingestion. Standardizing to GeoParquet brings a lot of value that addresses all of the aspects of FAIR data and value pyramid.

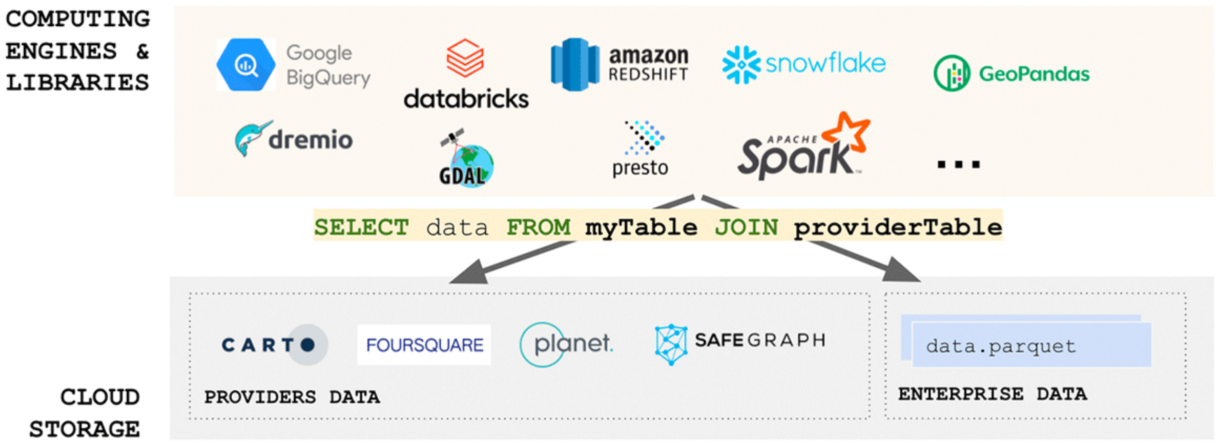


Figure 6: Geoparquet as a geospatial standard data format

Why introduce another format into an already complex ecosystem? GeoParquet isn't a new format — it is a schema specification for Apache Parquet format that is already widely adopted and used by the industry and the community. Parquet as the base format supports binary columns and allows for storage of arbitrary data payload. At the same time the format supports structured data columns that can store metadata together with the data payload. This makes it a choice that promotes interoperability and reproducibility. Finally, Delta Lake format has been built on top of parquet and brings ACID properties to the table. ACID properties of a format are crucial for reproducibility and for trusted outputs. In addition, Delta is the format used by scalable data sharing solution Delta Sharing.

Delta Sharing enables enterprise scale data sharing between any public cloud using Databricks (DIY options for private cloud are available using open source building blocks). Delta Sharing completely abstracts the need for custom built Rest APIs for exposing data to other third parties. Any data asset stored in Delta (using GeoParquet schema) automatically becomes a data product that can be exposed to external parties in a controlled and governed manner. Delta Sharing has been built from the ground up with security best practices in mind.

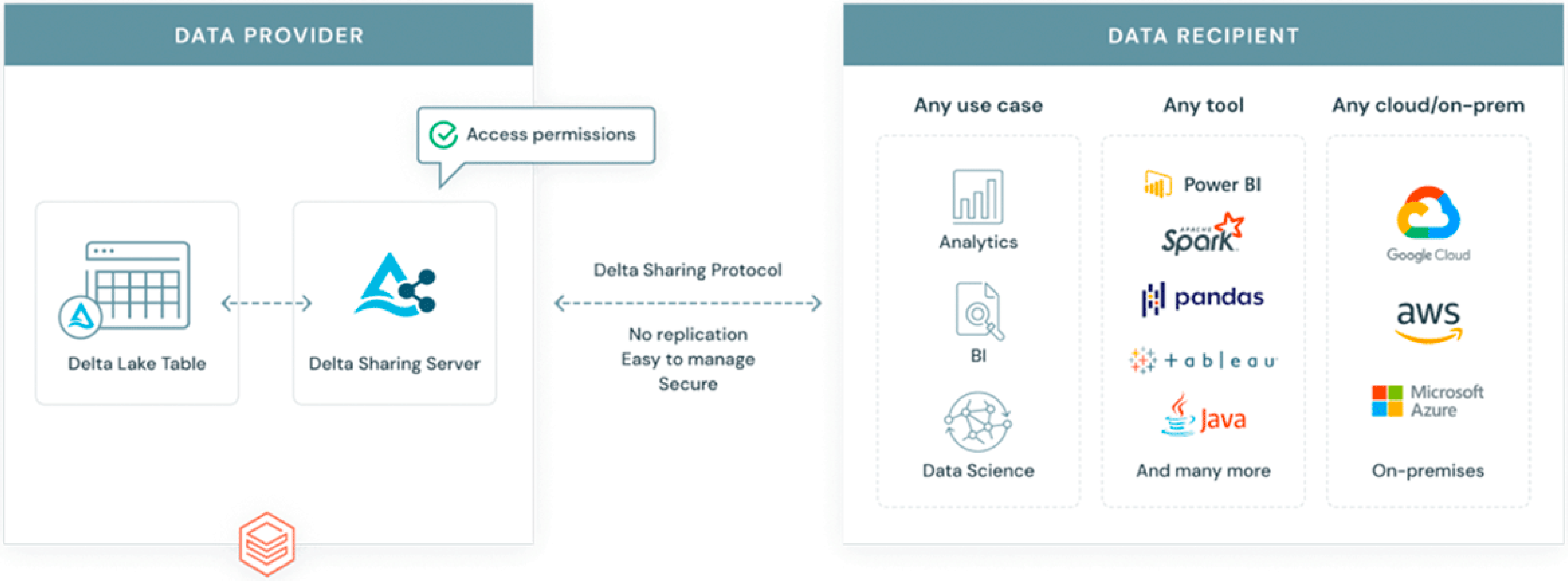


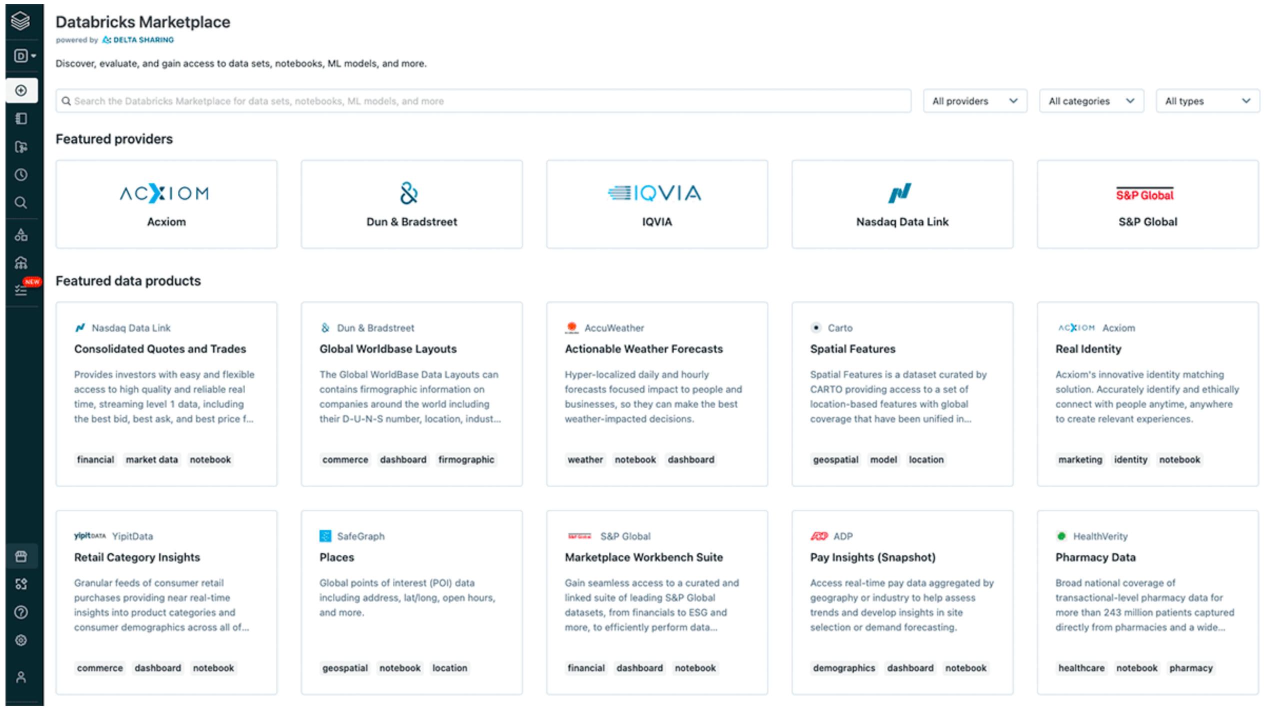
Figure 7: Delta Sharing simplifying data access in the ecosystem

## Circular data economy

Borrowing the concepts from the sustainability domain, we can define a circular data economy as a system in which data is collected, shared, and used in a way that maximizes its value while minimizing waste and negative impacts, such as unnecessary compute time, untrustworthy insights, or biased actions based data pollutants. Reusability is the key concept in this consideration — how can we minimize the "reinvention of the wheel." There are countless data assets out in the wild that represent the same area, same concepts with just ever slight alterations to better match a specific use case. Is this due to the actual optimizations or due to the fact it was easier to create a new copy of the assets than to reuse the existing ones? Or was it too hard to find the existing data assets, or maybe it was too complex to define data access patterns.

Data asset duplication has many negative aspects in both FAIR considerations and data value pyramid considerations — having many disparate similar (but different) data assets that represent the same area and same concepts can deteriorate simplicity considerations of the data domain — it becomes hard to identify the data asset we actually can trust. It can also have very negative implications toward habit generation. Many niche communities will emerge that will standardize to themselves ignoring the best practices of the wider ecosystem, or worse yet they will not standardize at all.

In a circular data economy, data is treated as a valuable resource that can be used to create new products and services, as well as improving existing ones. This approach encourages the reuse and recycling of data, rather than treating it as a disposable commodity. Once again, we are using the sustainability analogy in a literal sense — we argue that this is the correct way of approaching the problem. Data pollutants are a real challenge for organizations both internally and externally. An article by The Guardian states that less than 1% of collected data is actually analyzed. There is too much data duplication, the majority of data is hard to access and deriving actual value is too cumbersome. Circular data economy promotes best practices and reusability of existing data assets allowing for a more consistent interpretation and insights across the wider data ecosystem.



Databricks Marketplace

powered by DELTA SHARING

Discover, evaluate, and gain access to data sets, notebooks, ML models, and more.

Search the Databricks Marketplace for data sets, notebooks, ML models, and more

All providers | All categories | All types

Featured providers

Acxiom | Dun & Bradstreet | IQVIA | Nasdaq Data Link | S&P Global

Featured data products

Nasdaq Data Link
Consolidated Quotes and Trades
Provides investors with easy and flexible access to high quality and reliable real time, streaming level 1 data, including the best bid, best ask, and best price f...
financial | market data | notebook

Dun & Bradstreet
Global Worldbase Layouts
The Global WorldBase Data Layouts can contains firmographic information on companies around the world including their D-U-N-S number, location, indust...
commerce | dashboard | firmographic

AccuWeather
Actionable Weather Forecasts
Hyper-localized daily and hourly forecasts focused impact to people and businesses, so they can make the best weather-impacted decisions.
weather | notebook | dashboard

Carto
Spatial Features
Spatial Features is a dataset curated by CARTO providing access to a set of location-based features with global coverage that have been unified in...
geospatial | model | location

Acxiom
Real Identity
Acxiom's innovative identity matching solution. Accurately identify and ethically connect with people anytime, anywhere to create relevant experiences.
marketing | identity | notebook

YipitData
Retail Category Insights
Granular feeds of consumer retail purchases providing near real-time insights into product categories and consumer demographics across all of...
commerce | dashboard | notebook

SafeGraph
Places
Global points of interest (POI) data including address, lat/long, open hours, and more.
geospatial | notebook | location

S&P Global
Marketplace Workbench Suite
Gain seamless access to a curated and linked suite of leading S&P Global datasets, from financials to ESG and more, to efficiently perform data...
financial | dashboard | notebook

ADP
Pay Insights (Snapshot)
Access real-time pay data aggregated by geography or industry to help assess trends and develop insights in site selection or demand forecasting.
demographics | dashboard | notebook

HealthVerity
Pharmacy Data
Broad national coverage of transactional-level pharmacy data for more than 243 million patients captured directly from pharmacies and a wide...
healthcare | notebook | pharmacy

Figure 8: Databricks Marketplace

Interoperability is a key component of FAIR data principles, and from interoperability a question of circularity comes to mind. How can we design an ecosystem that maximizes data utilization and data reuse? Once again, FAIR together with the value pyramid holds answers. Findability of the data is key to the data reuse and to solving for data pollution. With data assets that can be discovered easily we can avoid the recreation of same data assets in multiple places with just slight alteration. Instead we gain a coherent data ecosystem with data that can be easily combined and reused. Databricks has recently announced the Databricks Marketplace. The idea behind the marketplace is in line with the original definition of data product by DJ Patel. The marketplace will support sharing of data sets, notebooks, dashboards, and machine learning models. The critical building block for such a marketplace is the concept of Delta Sharing — the scalable, flexible and robust channel for sharing any data — geospatial data included.

Designing scalable data products that will live in the marketplace is crucial. In order to maximize the value add of each data product one should strongly consider FAIR principles and the product value pyramid. Without these guiding principles we will only increase the issues that are already present in the current systems. Each data product should solve a unique problem and should solve it in a simple, reproducible and robust way.

**You can read more on how Databricks Lakehouse Platform can help you accelerate time to value from your data products in the eBook: A New Approach to Data Sharing.**

**Start experimenting with these free Databricks notebooks.**

SECTION 2.7

# Data Lineage With Unity Catalog

by **PAUL ROOME, TAO FENG AND SACHIN THAKUR**

June 8, 2022

This blog will discuss the importance of data lineage, some of the common use cases, our vision for better data transparency and data understanding with data lineage.

## What is data lineage and why is it important?

Data lineage describes the transformations and refinements of data from source to insight. Lineage includes capturing all the relevant metadata and events associated with the data in its lifecycle, including the source of the data set, what other data sets were used to create it, who created it and when, what transformations were performed, what other data sets leverage it, and many other events and attributes. With a data lineage solution, data teams get an end-to-end view of how data is transformed and how it flows across their data estate.

As more and more organizations embrace a data-driven culture and set up processes and tools to democratize and scale data and AI, data lineage is becoming an essential pillar of a pragmatic data management and governance strategy.

To understand the importance of data lineage, we have highlighted some of the common use cases we have heard from our customers below.

### Impact analysis

Data goes through multiple updates or revisions over its lifecycle, and understanding the potential impact of any data changes on downstream consumers becomes important from a risk management standpoint. With data lineage, data teams can see all the downstream consumers — applications, dashboards, machine learning models or data sets, etc. — impacted by data changes, understand the severity of the impact, and notify the relevant stakeholders. Lineage also helps IT teams proactively communicate data migrations to the appropriate teams, ensuring business continuity.

### Data understanding and transparency

Organizations deal with an influx of data from multiple sources, and building a better understanding of the context around data is paramount to ensure the trustworthiness of the data. Data lineage is a powerful tool that enables data leaders to drive better transparency and understanding of data in their organizations. Data lineage also empowers data consumers such as data scientists, data engineers and data analysts to be context-aware as they perform analyses, resulting in better quality outcomes. Finally, data stewards can see which data sets are no longer accessed or have become obsolete to retire unnecessary data and ensure data quality for end business users .

### Debugging and diagnostics

You can have all the checks and balances in place, but something will eventually break. Data lineage helps data teams perform a root cause analysis of any errors in their data pipelines, applications, dashboards, machine learning models, etc., by tracing the error to its source. This significantly reduces the debugging time, saving days, or in many cases, months of manual effort.

### Compliance and audit readiness

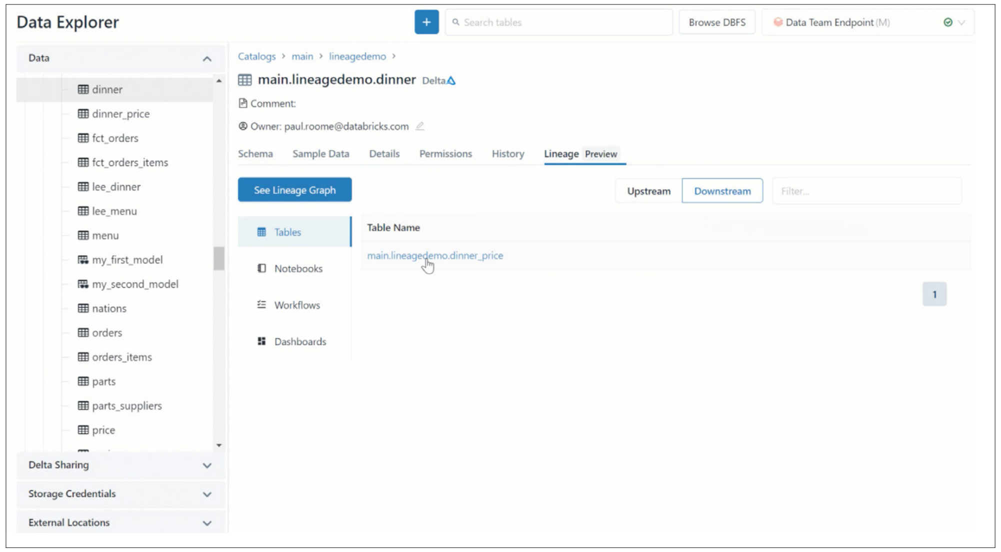
Many compliance regulations, such as the General Data Protection Regulation (GDPR), California Consumer Privacy Act (CCPA), Health Insurance Portability and Accountability Act (HIPPA), Basel Committee on Banking Supervision (BCBS) 239, and Sarbanes-Oxley Act (SOX), require organizations to have clear understanding and visibility of data flow. As a result, data traceability becomes a key requirement in order for their data architecture to meet legal regulations. Data lineage helps organizations be compliant and audit-ready, thereby alleviating the operational overhead of manually creating the trails of data flows for audit reporting purposes.

## Effortless transparency and proactive control with data lineage

The lakehouse provides a pragmatic data management architecture that substantially simplifies enterprise data infrastructure and accelerates innovation by unifying your data warehousing and AI use cases on a single platform.
We believe data lineage is a key enabler of better data transparency and data understanding in your lakehouse, surfacing the relationships between data, jobs, and consumers, and helping organizations move toward proactive data management practices. For example:

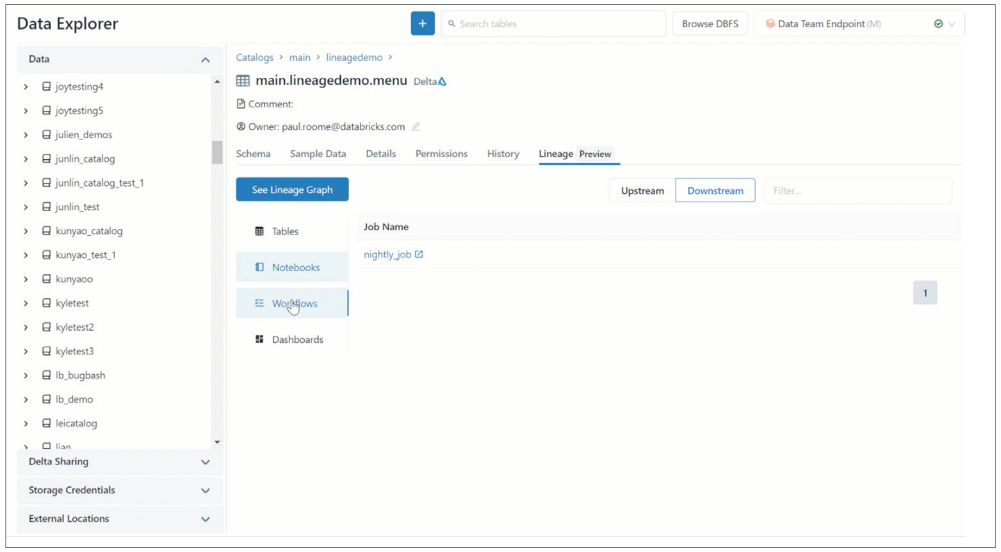
- As the owner of a dashboard, do you want to be notified next time that a table your dashboard depends upon wasn't loaded correctly?
- As a machine learning practitioner developing a model, do you want to be alerted that a critical feature in your model will be deprecated soon?
- As a governance admin, do you want to automatically control access to data based on its provenance?

All of these capabilities rely upon the automatic collection of data lineage across all use cases and personas — which is why the lakehouse and data lineage are a powerful combination.



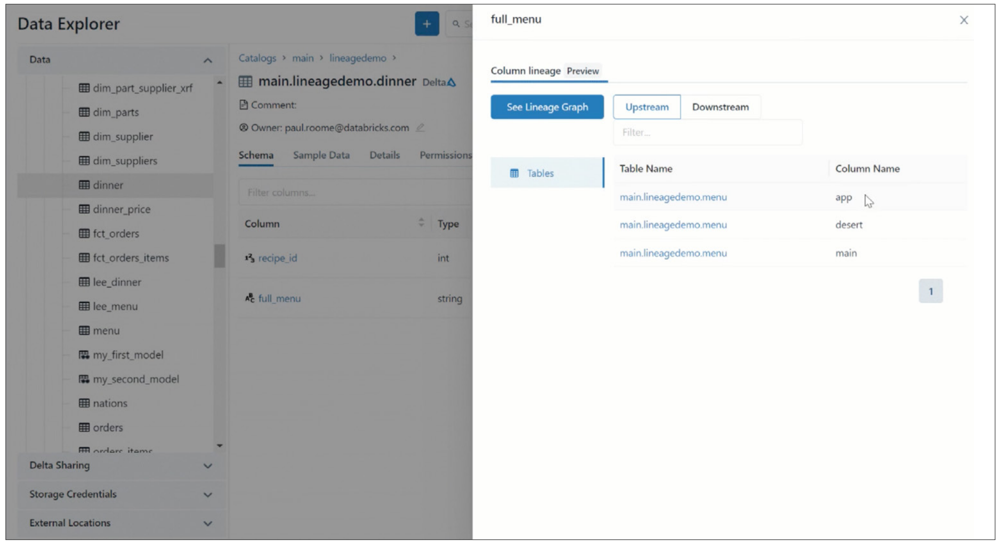
| Table Name |
|---|
| main.lineagedemo.dinner_price |

Data lineage for tables



| Job Name |
|---|
| nightly_job |

Data Lineage for notebooks, workflows, dashboards



| Table Name | Column Name |
|---|---|
| main.lineagedemo.menu | app |
| main.lineagedemo.menu | desert |
| main.lineagedemo.menu | main |

Data lineage for table columns

**Built-in security:** Lineage graphs in Unity Catalog are privilege-aware and share the same permission model as Unity Catalog. If users do not have access to a table, they will not be able to explore the lineage associated with the table, adding an additional layer of security for privacy considerations.

**Easily exportable via REST API:** Lineage can be visualized in the Data Explorer in near real-time, and retrieved via REST API to support integrations with our catalog partners.

## Getting started with data lineage in Unity Catalog

Data lineage is available with Databricks Premium and Enterprise tiers for no additional cost. If you already are a Databricks customer, follow the data lineage guides (AWS | Azure) to get started. If you are not an existing Databricks customer, sign up for a free trial with a Premium or Enterprise workspace.

SECTION 2.8

# Easy Ingestion to Lakehouse With COPY INTO

by AEMRO AMARE, EMMA LIU, AMIT KARA and JASRAJ DANGE

January 17, 2023

A new data management architecture known as the data lakehouse emerged independently across many organizations and use cases to support AI and BI directly on vast amounts of data. One of the key success factors for using the data lakehouse for analytics and machine learning is the ability to quickly and easily ingest data of various types, including data from on-premises storage platforms (data warehouses, mainframes), real-time streaming data, and bulk data assets.

As data ingestion into the lakehouse is an ongoing process that feeds the proverbial ETL pipeline, you will need multiple options to ingest various formats, types and latency of data. For data stored in cloud object stores such as AWS S3, Google Cloud Storage and Azure Data Lake Storage, Databricks offers Auto Loader, a natively integrated feature, that allows data engineers to ingest millions of files from the cloud storage continuously. In other streaming cases (e.g., IoT sensor or clickstream data), Databricks provides native connectors for Apache Spark Structured Streaming to quickly ingest data from popular message queues, such as Apache Kafka, Azure Event Hubs or AWS Kinesis at low latencies. Furthermore, many customers can leverage popular ingestion tools that integrate with Databricks, such as Fivetran — to easily ingest data from enterprise applications, databases, mainframes and more into the lakehouse. Finally, analysts can use the simple "COPY INTO" command to pull new data into the lakehouse automatically, without the need to keep track of which files have already been processed.

This blog focuses on COPY INTO, a simple yet powerful SQL command that allows you to perform batch file ingestion into Delta Lake from cloud object stores. It's idempotent, which guarantees to ingest files with exactly-once semantics when executed multiple times, supporting incremental appends and simple transformations. It can be run once, in an ad hoc manner, and can be scheduled through Databricks Workflows. In recent Databricks Runtime releases, COPY INTO introduced new functionalities for data preview, validation, enhanced error handling, and a new way to copy into a schemaless Delta Lake table so that users can get started quickly, completing the end-to-end user journey to ingest from cloud object stores. Let's take a look at the popular COPY INTO use cases.

## 1. Ingesting data for the first time

COPY INTO requires a table to exist as it ingests the data into a target Delta table. However, you have no idea what your data looks like. You first create an empty Delta table.

```
CREATE TABLE my_example_data;
```

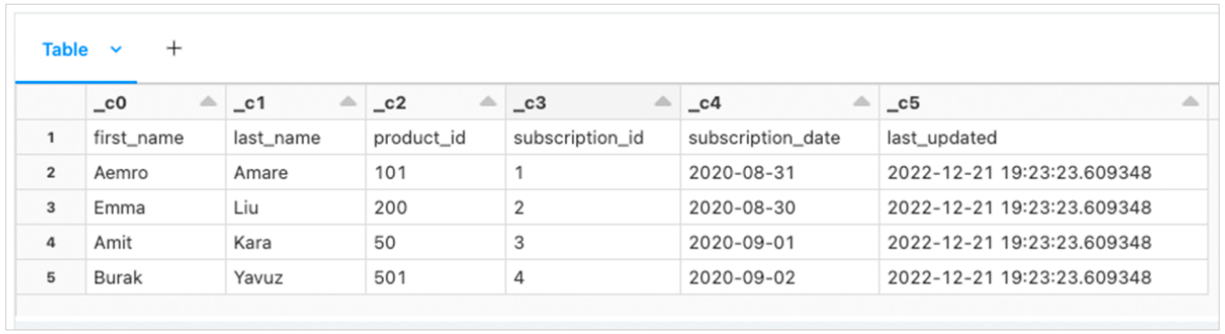
Before you write out your data, you may want to preview it and ensure the data looks correct. The COPY INTO Validate mode is a new feature in Databricks Runtime 10.3 and above that allows you to preview and validate source data before ingesting many files from the cloud object stores. These validations include:

- if the data can be parsed
- the schema matches that of the target table or if the schema needs to be evolved
- all nullability and check constraints on the table are met

```
COPY INTO my_example_data
FROM 's3://my-bucket/exampleData'
FILEFORMAT = CSV
VALIDATE
COPY_OPTIONS ('mergeSchema' = 'true')
```

The default for data validation is to parse all the data in the source directory to ensure that there aren't any issues, but the rows returned for preview are limited. Optionally, you can provide the number of rows to preview after VALIDATE.

The COPY_OPTION "mergeSchema" specifies that it is okay to evolve the schema of your target Delta table. Schema evolution only allows the addition of new columns, and does not support data type changes for existing columns. In other use cases, you can omit this option if you intend to manage your table schema more strictly as your data pipeline may have strict schema requirements and may not want to evolve the schema at all times. However, our target Delta table in the example above is an empty, columnless table at the moment; therefore, we have to specify the COPY_OPTION "mergeSchema" here.



| | _c0 | _c1 | _c2 | _c3 | _c4 | _c5 |
|---|---|---|---|---|---|---|
| 1 | first_name | last_name | product_id | subscription_id | subscription_date | last_updated |
| 2 | Aemro | Amare | 101 | 1 | 2020-08-31 | 2022-12-21 19:23:23.609348 |
| 3 | Emma | Liu | 200 | 2 | 2020-08-30 | 2022-12-21 19:23:23.609348 |
| 4 | Amit | Kara | 50 | 3 | 2020-09-01 | 2022-12-21 19:23:23.609348 |
| 5 | Burak | Yavuz | 501 | 4 | 2020-09-02 | 2022-12-21 19:23:23.609348 |

Figure 1: COPY INTO VALIDATE mode output

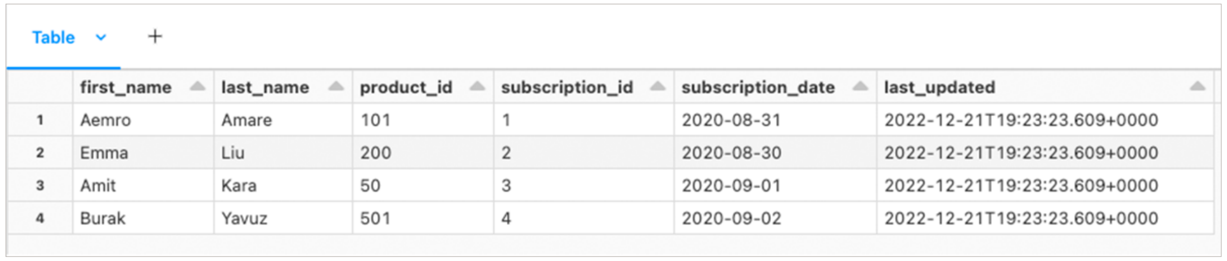
## 2. Configuring COPY INTO

When looking over the results of VALIDATE (see Figure 1), you may notice that your data doesn't look like what you want. Aren't you glad you previewed your data set first? The first thing you notice is the column names are not what is specified in the CSV header. What's worse, the header is shown as a row in your data. You can configure the CSV parser by specifying FORMAT_OPTIONS.
Let's add those next.

```
COPY INTO my_example_data
FROM 's3://my-bucket/exampleData'
FILEFORMAT = CSV
VALIDATE
FORMAT_OPTIONS ('header' = 'true', 'inferSchema' = 'true', 'mergeSchema' =
'true')
COPY_OPTIONS ('mergeSchema' = 'true')
```

When using the FORMAT OPTION, you can tell COPY INTO to infer the data types of the CSV file by specifying the inferSchema option; otherwise, all default data types are STRINGs. On the other hand, binary file formats like AVRO and PARQUET do not need this option since they define their own schema. Another option, "mergeSchema" states that the schema should be inferred over a comprehensive sample of CSV files rather than just one. The comprehensive list of format-specific options can be found in the documentation.

Figure 2 shows the validate output that the header is properly parsed.



| | first_name | last_name | product_id | subscription_id | subscription_date | last_updated |
|---|---|---|---|---|---|---|
| 1 | Aemro | Amare | 101 | 1 | 2020-08-31 | 2022-12-21T19:23:23.609+0000 |
| 2 | Emma | Liu | 200 | 2 | 2020-08-30 | 2022-12-21T19:23:23.609+0000 |
| 3 | Amit | Kara | 50 | 3 | 2020-09-01 | 2022-12-21T19:23:23.609+0000 |
| 4 | Burak | Yavuz | 501 | 4 | 2020-09-02 | 2022-12-21T19:23:23.609+0000 |

Figure 2: COPY INTO VALIDATE mode output with enabled header and inferSchema

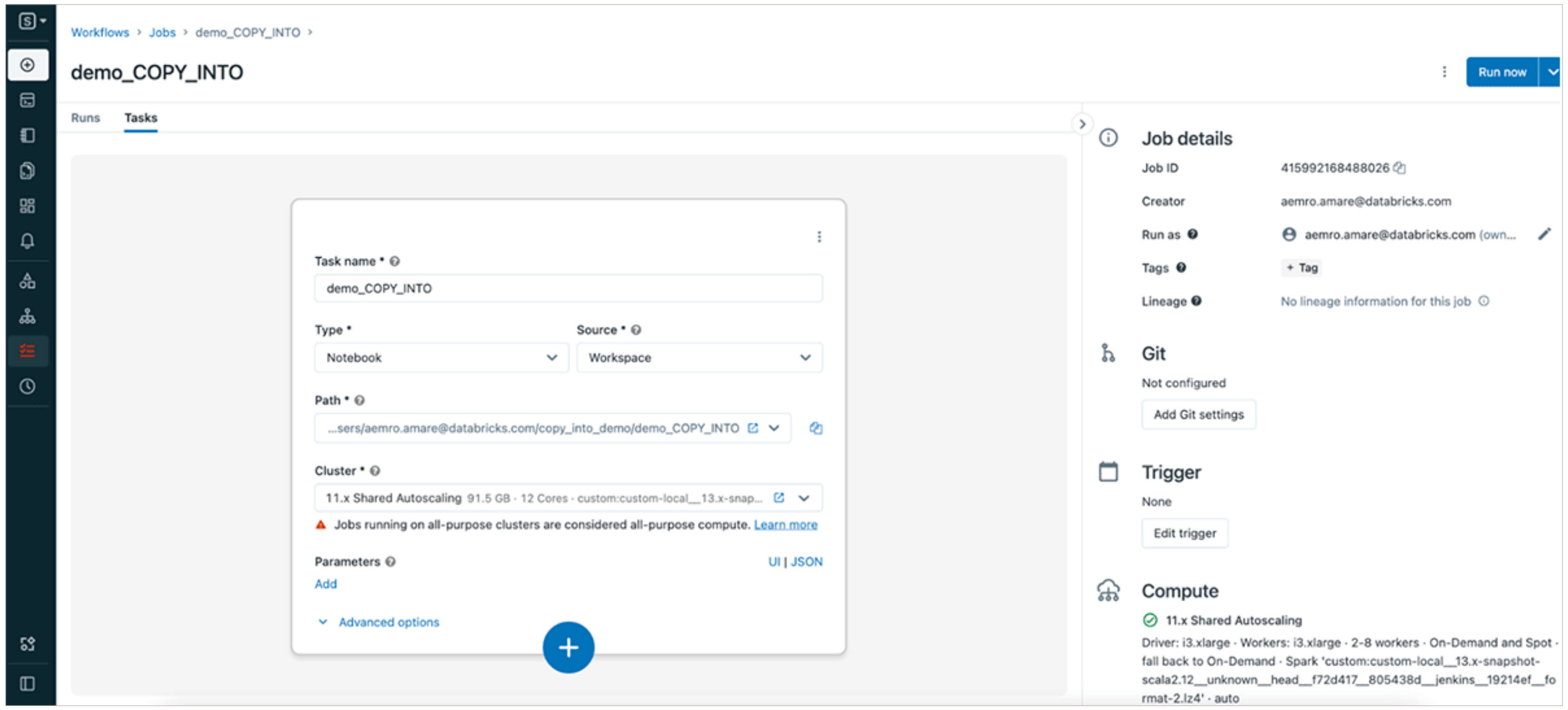
## 3. Appending data to a Delta table

Now that the preview looks good, we can remove the VALIDATE keyword and execute the COPY INTO command.

```
COPY INTO my_example_data
FROM 's3://my-bucket/exampleData'
FILEFORMAT = CSV
FORMAT_OPTIONS ('header' = 'true', 'inferSchema' = 'true', 'mergeSchema' =
'true')
COPY_OPTIONS ('mergeSchema' = 'true')
```

COPY INTO keeps track of the state of files that have been ingested. Unlike commands like INSERT INTO, users get idempotency with COPY INTO, which means users won't get duplicate data in the target table when running COPY INTO multiple times from the same source data.

COPY INTO can be run once, in an ad hoc manner, and can be scheduled with Databricks Workflows. While COPY INTO does not support low latencies for ingesting natively, you can trigger COPY INTO through orchestrators like Apache Airflow.



demo_COPY_INTO

Task name * demo_COPY_INTO

Type * Notebook

Source * Workspace

Path * ...sers/aemro.amare@databricks.com/copy_into_demo/demo_COPY_INTO

Cluster * 11.x Shared Autoscaling 91.5 GB · 12 Cores · custom:custom-local__13.x-snap...

Jobs running on all-purpose clusters are considered all-purpose compute. Learn more

Parameters UI | JSON

Job details

Job ID 415992168488026

Creator aemro.amare@databricks.com

Run as aemro.amare@databricks.com (own...

Tags

Lineage No lineage information for this job

Git

Not configured

Trigger

None

Compute

11.x Shared Autoscaling

Driver: i3.xlarge · Workers: i3.xlarge · 2-8 workers · On-Demand and Spot · fall back to On-Demand · Spark 'custom:custom-local__13.x-snapshot-scala2.12__unknown__head__f72d417__805438d__jenkins__19214ef__format-2.lz4' · auto

Figure 3: Databricks workflow UI to schedule a task

## 4. Secure data access with COPY INTO

COPY INTO supports secure access in several ways. In this section, we want to highlight two new options you can use in both Databricks SQL and notebooks from recent releases:

### Unity Catalog

With the general availability of Databrick Unity Catalog, you can use COPY INTO to ingest data to Unity Catalog managed or external tables from any source and file format supported by COPY INTO. Unity Catalog also adds new options for configuring secure access to raw data, allowing you to use Unity Catalog external locations or storage credentials to access data in cloud object storage. Learn more about how to use COPY INTO with Unity Catalog.

### Temporary Credentials

What if you have not configured Unity Catalog or instance profile? How about data from a trusted third party bucket? Here is a convenient COPY INTO feature that allows you to ingest data with inline temporary credentials to handle the ad hoc bulk ingestion use case.

```
COPY INTO my_example_data
FROM 's3://my-bucket/exampleDataPath' WITH (
  CREDENTIAL (AWS_ACCESS_KEY = '...', AWS_SECRET_KEY = '...', AWS_SESSION_
TOKEN = '...')
)
FILEFORMAT = CSV
```

## 5. Filtering files for ingestion

What about ingesting a subset of files where the filenames match a pattern? You can apply glob patterns — a glob pattern that identifies the files to load from the source directory. For example, let's filter and ingest files which contain the word `raw_data` in the filename below.

```
COPY INTO my_example_data
  FROM  's3://my-bucket/exampleDataPath'
  FILEFORMAT = CSV
  PATTERN = '*raw_data*.csv'
  FORMAT_OPTIONS ('header' = 'true')
```

## 6. Ingest files in a time period

In data engineering, it is frequently necessary to ingest files that have been modified before or after a specific timestamp. Data between two timestamps may also be of interest. The 'modifiedAfter' and 'modifiedBefore' format options offered by COPY INTO allow users to ingest data from a chosen time window into a Delta table.

```
COPY INTO my_example_data
  FROM  's3://my-bucket/exampleDataPath'
  FILEFORMAT = CSV
  PATTERN = '*raw_data_*.csv'
  FORMAT_OPTIONS('header' = 'true', 'modifiedAfter' = '2022-09-
12T10:53:11.000+0000')
```

## 7. Correcting data with the force option

Because COPY INTO is by default idempotent, running the same query against the same source files more than once has no effect on the destination table after the initial execution. You must propagate changes to the target table because, in real-world circumstances, source data files in cloud object storage may be altered for correction at a later time. In such a case, it is possible to first erase the data from the target table before ingesting the more recent data files from the source. For this operation you only need to set the copy option ‘force’ to ‘true’.

```
COPY INTO my_example_data
  FROM  's3://my-bucket/exampleDataPath'
  FILEFORMAT = CSV
  PATTERN = '*raw_data_2022*.csv'
  FORMAT_OPTIONS('header' = 'true')
  COPY_OPTIONS ('force' = 'true')
```

## 8. Applying simple transformations

What if you want to rename columns? Or the source data has changed and a previous column has been renamed to something else? You don’t want to ingest that data as two separate columns, but as a single column. We can leverage the SELECT statement in COPY INTO perform simple transformations.

```
COPY INTO demo.my_example_data
  FROM ( SELECT  concat(first_name, " ", last_name) as full_name,
                 * EXCEPT (first_name, last_name)
    FROM 's3://my-bucket/exampleDataPath'
  )
  FILEFORMAT = CSV
  PATTERN = '*.csv'
  FORMAT_OPTIONS('header' = 'true')
  COPY_OPTIONS ('force' = 'true')
```

## 9. Error handling and observability with COPY INTO

### Error handling:

How about ingesting data with file corruption issues? Common examples of file corruption are:

- Files with an incorrect file format
- Failure to decompress
- Unreadable files (e.g., invalid Parquet)

COPY INTO's format option ignoreCorruptFiles helps skip those files while processing. The result of the COPY INTO command returns the number of files skipped in the num_skipped_corrupt_files column. In addition, these corrupt files aren't tracked by the ingestion state in COPY INTO, therefore they can be reloaded in a subsequent execution once the corruption is fixed. This option is available in Databricks Runtime 11.0+.

You can see which files have been detected as corrupt by running COPY INTO in VALIDATE mode.

```
COPY INTO my_example_data
  FROM  's3://my-bucket/exampleDataPath'
  FILEFORMAT = CSV
  VALIDATE ALL
  FORMAT_OPTIONS('ignoreCorruptFiles' = 'true')
```

**Observability:**
In Databricks Runtime 10.5, file metadata column was introduced to provide input file metadata information, which allows users to monitor and get key properties of the ingested files like path, name, size and modification time, by querying a hidden STRUCT column called _metadata. To include this information in the destination, you must explicitly reference the _metadata column in your query in COPY INTO.

```
COPY INTO my_example_data
  FROM (
     SELECT *, _metadata source_metadata FROM 's3://my-bucket/
exampleDataPath'
  )
  FILEFORMAT = CSV
```

## How does it compare to Auto Loader?

COPY INTO is a simple and powerful command to use when your source directory contains a small number of files (i.e., thousands of files or less), and if you prefer SQL. In addition, COPY INTO can be used over JDBC to push data into Delta Lake at your convenience, a common pattern by many ingestion partners. To ingest a larger number of files both in streaming and batch we recommend using Auto Loader. In addition, for a modern data pipeline based on medallion architecture, we recommend using Auto Loader in Delta Live Tables pipelines, leveraging advanced capabilities of automatic error handling, quality control, data lineage and setting expectations in a declarative approach.

## How to get started?

To get started, you can go to **Databricks SQL** query editor, update and run the example SQL commands to ingest from your cloud object stores. Check out the options in No. 4 to establish secure access to your data for querying it in Databricks SQL. To get familiar with COPY INTO in Databricks SQL, you can also follow this quickstart tutorial.

As an alternative, you can use this notebook in Data Science & Engineering and Machine Learning workspaces to learn most of the COPY INTO features in this blog, where source data and target Delta tables are generated in DBFS.

More tutorials for COPY INTO can be found here.

SECTION 2.9

# Simplifying Change Data Capture With Databricks Delta Live Tables

by MOJGAN MAZOUCHI

April 25, 2022

This guide will demonstrate how you can leverage change data capture in Delta Live Tables pipelines to identify new records and capture changes made to the data set in your data lake. Delta Live Tables pipelines enable you to develop scalable, reliable and low latency data pipelines, while performing change data capturee in your data lake with minimum required computation resources and seamless out-of-order data handling.

> **Note:** We recommend following Getting Started with Delta Live Tables which explains creating scalable and reliable pipelines using Delta Live Tables (DLT) and its declarative ETL definitions.

## Background on change data capture

Change data capture (CDC) is a process that identifies and captures incremental changes (data deletes, inserts and updates) in databases, like tracking customer, order or product status for near-real-time data applications. CDC provides real-time data evolution by processing data in a continuous incremental fashion as new events occur.

Since over 80% of organizations plan on implementing multicloud strategies by 2025, choosing the right approach for your business that allows seamless real-time centralization of all data changes in your ETL pipeline across multiple environments is critical.

By capturing CDC events, Databricks users can re-materialize the source table as Delta Table in Lakehouse and run their analysis on top of it, while being able to combine data with external systems. The MERGE INTO command in Delta Lake on Databricks enables customers to efficiently upsert and delete records in their data lakes — you can check out our previous deep dive on the topic here. This is a common use case that we observe many of Databricks customers are leveraging Delta Lakes to perform, and keeping their data lakes up to date with real-time business data.

While Delta Lake provides a complete solution for real-time CDC synchronization in a data lake, we are now excited to announce the change data capture feature in Delta Live Tables that makes your architecture even simpler, more efficient and scalable. DLT allows users to ingest CDC data seamlessly using SQL and Python.

Earlier CDC solutions with Delta tables were using MERGE INTO operation, which requires manually ordering the data to avoid failure when multiple rows of the source data set match while attempting to update the same rows of the target

Delta table. To handle the out-of-order data, there was an extra step required to preprocess the source table using a foreachBatch implementation to eliminate the possibility of multiple matches, retaining only the latest change for each key (see the change data capture example). The new APPLY CHANGES INTO operation in DLT pipelines automatically and seamlessly handles out-of-order data without any need for data engineering manual intervention.

## CDC with Databricks Delta Live Tables

In this blog, we will demonstrate how to use the APPLY CHANGES INTO command in Delta Live Tables pipelines for a common CDC use case where the CDC data is coming from an external system. A variety of CDC tools are available such as Debezium, Fivetran, Qlik Replicate, Talend, and StreamSets. While specific implementations differ, these tools generally capture and record the history of data changes in logs; downstream applications consume these CDC logs. In our example, data is landed in cloud object storage from a CDC tool such as Debezium, Fivetran, etc.

We have data from various CDC tools landing in a cloud object storage or a message queue like Apache Kafka. Typically we see CDC used in an ingestion to what we refer as the medallion architecture. A medallion architecture is a data design pattern used to logically organize data in a Lakehouse, with the goal of incrementally and progressively improving the structure and quality of data as it flows through each layer of the architecture. Delta Live Tables allows you to seamlessly apply changes from CDC feeds to tables in your Lakehouse; combining this functionality with the medallion architecture allows for incremental changes to easily flow through analytical workloads at scale. Using CDC together with the medallion architecture provides multiple benefits to users since only changed or added data needs to be processed. Thus, it enables users to cost-effectively keep Gold tables up-to-date with the latest business data.

> **NOTE:** The example here applies to both SQL and Python versions of CDC and also on a specific way to use the operations; to evaluate variations, please see the official documentation here.

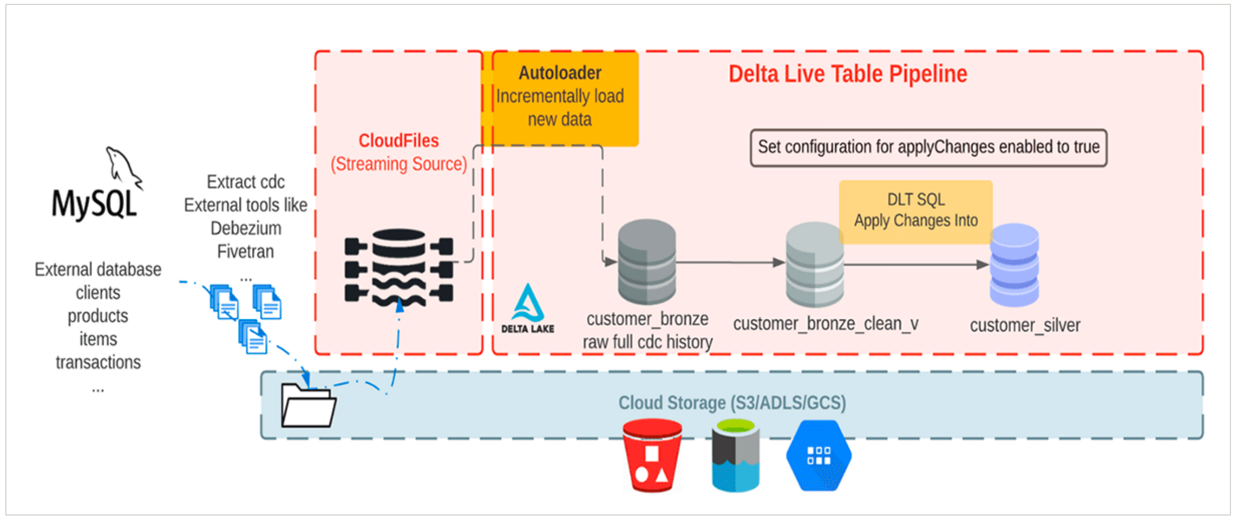
### Prerequisites

To get the most out of this guide, you should have a basic familiarity with:

- SQL or Python
- Delta Live Tables
- Developing ETL pipelines and/or working with Big Data systems
- Databricks interactive notebooks and clusters
- You must have access to a Databricks Workspace with permissions to create new clusters, run jobs, and save data to a location on external cloud object storage or DBFS
- For the pipeline we are creating in this blog, “Advanced” product edition which supports enforcement of data quality constraints, needs to be selected

## The data set

Here we are consuming realistic looking CDC data from an external database. In this pipeline, we will use the Faker library to generate the data set that a CDC tool like Debezium can produce and bring into cloud storage for the initial ingest in Databricks. Using Auto Loader we incrementally load the messages from cloud object storage, and store them in the Bronze table as it stores the raw messages. The Bronze tables are intended for data ingestion which enable quick access to a single source of truth. Next we perform APPLY CHANGES INTO from the cleaned Bronze layer table to propagate the updates downstream to the Silver table. As data flows to Silver tables, generally it becomes more refined and optimized ("just-enough") to provide an enterprise a view of all its key business entities. See the diagram below.



This blog focuses on a simple example that requires a JSON message with four fields of customer's name, email, address and id along with the two fields: operation (which stores operation code (DELETE, APPEND, UPDATE, CREATE) and operation_date (which stores the date and timestamp for the record came for each operation action) to describe the changed data.

To generate a sample data set with the above fields, we are using a Python package that generates fake data, Faker. You can find the notebook related to this data generation section here. In this notebook we provide the name and storage location to write the generated data there. We are using the DBFS functionality of Databricks; see the DBFS documentation to learn more about how it works. Then, we use a PySpark user-defined function to generate the synthetic data set for each field, and write the data back to the defined storage location, which we will refer to in other notebooks for accessing the synthetic data set.

## Ingesting the raw data set using Auto Loader

According to the medallion architecture paradigm, the Bronze layer holds the most raw data quality. At this stage we can incrementally read new data using Auto Loader from a location in cloud storage. Here we are adding the path to our generated data set to the configuration section under pipeline settings, which allows us to load the source path as a variable. So now our configuration under pipeline settings looks like below:

```
"configuration": {
"source": "/tmp/demo/cdc_raw"
}
```

Then we load this configuration property in our notebooks.

Let's take a look at the Bronze table we will ingest, a. In SQL, and b. Using Python

### A. SQL

```
SET spark.source;
CREATE STREAMING LIVE TABLE customer_bronze
(
address string,
email string,
id string,
firstname string,
lastname string,
operation string,
operation_date string,
_rescued_data string
)
TBLPROPERTIES ("quality" = "bronze")
COMMENT "New customer data incrementally ingested from cloud object
storage landing zone"
AS
SELECT *
FROM cloud_files("${source}/customers", "json", map("cloudFiles.
inferColumnTypes", "true"));
```

### B. PYTHON

```
import dlt
from pyspark.sql.functions import *
from pyspark.sql.types import *

source = spark.conf.get("source")

@dlt.table(name="customer_bronze",
                  comment = "New customer data incrementally ingested from
cloud object storage landing zone",
  table_properties={
    "quality": "bronze"
  }
)
def customer_bronze():
  return (
    spark.readStream.format("cloudFiles") \
      .option("cloudFiles.format", "json") \
      .option("cloudFiles.inferColumnTypes", "true") \
      .load(f"{source}/customers")
  )
```

The above statements use the Auto Loader to create a streaming live table called customer_bronze from json files. When using Auto Loader in Delta Live Tables, you do not need to provide any location for schema or checkpoint, as those locations will be managed automatically by your DLT pipeline.

Auto Loader provides a Structured Streaming source called cloud_files in SQL and cloudFiles in Python, which takes a cloud storage path and format as parameters.

To reduce compute costs, we recommend running the DLT pipeline in Triggered mode as a micro-batch assuming you do not have very low latency requirements.

## Expectations and high-quality data

In the next step to create a high-quality, diverse, and accessible data set, we impose quality check expectation criteria using Constraints. Currently, a constraint can be either retain, drop, or fail. For more detail see here. All constraints are logged to enable streamlined quality monitoring.

### A. SQL

```
CREATE TEMPORARY STREAMING LIVE TABLE customer_bronze_clean_v(
  CONSTRAINT valid_id EXPECT (id IS NOT NULL) ON VIOLATION DROP ROW,
  CONSTRAINT valid_address EXPECT (address IS NOT NULL),
  CONSTRAINT valid_operation EXPECT (operation IS NOT NULL) ON VIOLATION
DROP ROW
)
TBLPROPERTIES ("quality" = "silver")
COMMENT "Cleansed bronze customer view (i.e. what will become Silver)"
AS SELECT *
FROM STREAM(LIVE.customer_bronze);
```

### B. PYTHON

```
@dlt.view(name="customer_bronze_clean_v",
  comment="Cleansed bronze customer view (i.e. what will become Silver)")

@dlt.expect_or_drop("valid_id", "id IS NOT NULL")
@dlt.expect("valid_address", "address IS NOT NULL")
@dlt.expect_or_drop("valid_operation", "operation IS NOT NULL")

def customer_bronze_clean_v():
  return dlt.read_stream("customer_bronze") \
            .select("address", "email", "id", "firstname", "lastname",
"operation", "operation_date", "_rescued_data")
```

## Using APPLY CHANGES INTO statement to propagate changes to downstream target table

Prior to executing the Apply Changes Into query, we must ensure that a target streaming table which we want to hold the most up-to-date data exists. If it does not exist we need to create one. Below cells are examples of creating a target streaming table. Note that at the time of publishing this blog, the target streaming table creation statement is required along with the Apply Changes Into query, and both need to be present in the pipeline — otherwise your table creation query will fail.

### A. SQL

```
CREATE STREAMING LIVE TABLE customer_silver
TBLPROPERTIES ("quality" = "silver")
COMMENT "Clean, merged customers";
```

### B. PYTHON

```
dlt.create_target_table(name="customer_silver",
  comment="Clean, merged customers",
  table_properties={
    "quality": "silver"
  }
)
```

Now that we have a target streaming table, we can propagate changes to the downstream target table using the Apply Changes Into query. While CDC feed comes with INSERT, UPDATE and DELETE events, DLT default behavior is to apply INSERT and UPDATE events from any record in the source data set matching on primary keys, and sequenced by a field which identifies the order of events. More specifically it updates any row in the existing target table that matches the primary key(s) or inserts a new row when a matching record does not exist in the target streaming table. We can use APPLY AS DELETE WHEN in SQL, or its equivalent apply_as_deletes argument in Python to handle DELETE events.

In this example we used "id" as my primary key, which uniquely identifies the customers and allows CDC events to apply to those identified customer records in the target streaming table. Since "operation_date" keeps the logical order of CDC events in the source data set, we use "SEQUENCE BY operation_date" in SQL, or its equivalent "sequence_by = col("operation_date")" in Python to handle change events that arrive out of order. Keep in mind that the field value we use with SEQUENCE BY (or sequence_by) should be unique among all updates to the same key. In most cases, the sequence by column will be a column with timestamp information.

Finally we used "COLUMNS * EXCEPT (operation, operation_date, _rescued_data)" in SQL, or its equivalent "except_column_list"= ["operation", "operation_date", "_rescued_data"] in Python to exclude three columns of "operation", "operation_date", "_rescued_data" from the target streaming table. By default all the columns are included in the target streaming table, when we do not specify the "COLUMNS" clause.

### A. SQL

```
APPLY CHANGES INTO LIVE.customer_silver
FROM stream(LIVE.customer_bronze_clean_v)
  KEYS (id)
  APPLY AS DELETE WHEN operation = "DELETE"
  SEQUENCE BY operation_date
  COLUMNS * EXCEPT (operation, operation_date,
_rescued_data);
```

### B. PYTHON

```
dlt.apply_changes(
  target = "customer_silver",
  source = "customer_bronze_clean_v",
  keys = ["id"],
  sequence_by = col("operation_date"),
  apply_as_deletes = expr("operation = 'DELETE'"),
  except_column_list = ["operation", "operation_date", "_rescued_data"])
```

To check out the full list of available clauses see here.

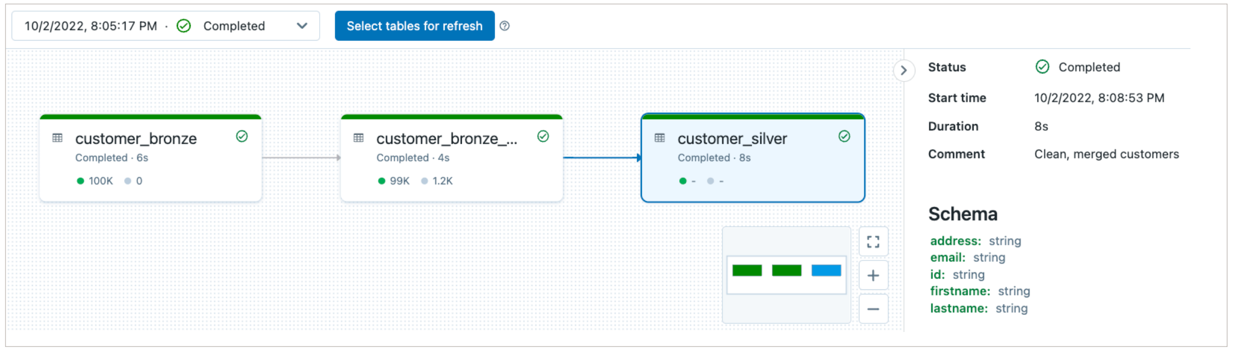
Please note that, at the time of publishing this blog, a table that reads from the target of an APPLY CHANGES INTO query or apply_changes function must be a live table, and cannot be a streaming live table.

A SQL and Python notebook is available for reference for this section. Now that we have all the cells ready, let’s create a pipeline to ingest data from cloud object storage. Open Jobs in a new tab or window in your workspace, and select “Delta Live Tables.”

The pipeline associated with this blog has the following DLT pipeline settings:

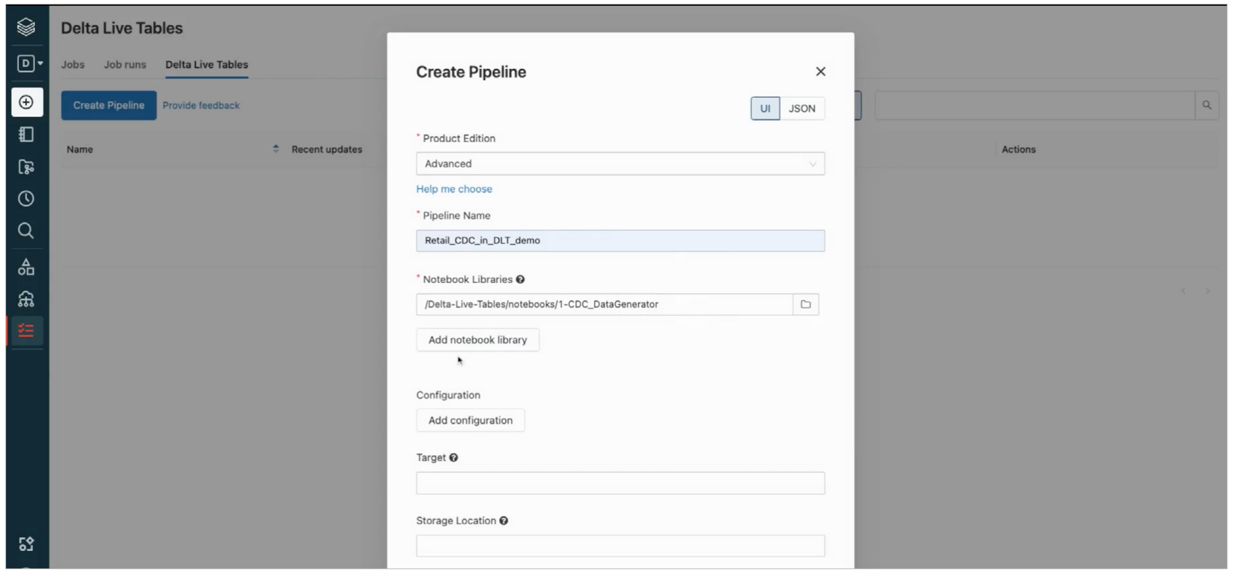
```
{
     "clusters": [
        {
            "label": "default",
            "num_workers": 1
        }
    ],
    "development": true,
    "continuous": false,
    "edition": "advanced",
    "photon": false,
    "libraries": [
        {
            "notebook": {
"path":"/Repos/mojgan.mazouchi@databricks.com/Delta-Live-Tables/
notebooks/1-CDC_DataGenerator"
            }
        },
        {
            "notebook": {
"path":"/Repos/mojgan.mazouchi@databricks.com/Delta-Live-Tables/
notebooks/2-Retail_DLT_CDC_sql"
            }
        }
    ],
    "name": "CDC_blog",
    "storage": "dbfs:/home/mydir/myDB/dlt_storage",
    "configuration": {
        "source": "/tmp/demo/cdc_raw",
        "pipelines.applyChangesPreviewEnabled": "true"
    },
    "target": "my_database"
}
```

1. Select “Create Pipeline” to create a new pipeline
2. Specify a name such as “Retail CDC Pipeline”
3. Specify the Notebook Paths that you already created earlier, one for the generated data set using Faker package, and another path for the ingestion of the generated data in DLT. The second notebook path can refer to the notebook written in SQL, or Python depending on your language of choice.
4. To access the data generated in the first notebook, add the data set path in configuration. Here we stored data in “/tmp/demo/cdc_raw/customers”, so we set “source” to “/tmp/demo/cdc_raw/” to reference “source/customers” in our second notebook.
5. Specify the Target (which is optional and referring to the target database), where you can query the resulting tables from your pipeline
6. Specify the Storage Location in your object storage (which is optional), to access your DLT produced data sets and metadata logs for your pipeline
7. Set Pipeline Mode to Triggered. In Triggered mode, DLT pipeline will consume new data in the source all at once, and once the processing is done it will terminate the compute resource automatically. You can toggle between Triggered and Continuous modes when editing your pipeline settings. Setting “continuous”: false in the JSON is equivalent to setting the pipeline to Triggered mode.
8. For this workload you can disable the autoscaling under Autopilot Options, and use only one worker cluster. For production workloads, we recommend enabling autoscaling and setting the maximum numbers of workers needed for cluster size.
9. Select “Start”
10. Your pipeline is created and running now!



Status Completed
Start time 10/2/2022, 8:08:53 PM
Duration 8s
Comment Clean, merged customers

Schema
address: string
email: string
id: string
firstname: string
lastname: string



Create Pipeline

* Product Edition: Advanced
* Pipeline Name: Retail_CDC_in_DLT_demo
* Notebook Libraries: /Delta-Live-Tables/notebooks/1-CDC_DataGenerator
Configuration
Target
Storage Location

## DLT pipeline lineage observability and data quality monitoring

All DLT pipeline logs are stored in the pipeline's storage location. You can specify your storage location only when you are creating your pipeline. Note that once the pipeline is created you can no longer modify storage location.

You can check out our previous deep dive on the topic here. Try this notebook to see pipeline observability and data quality monitoring on the example DLT pipeline associated with this blog.

## Conclusion

In this blog, we showed how we made it seamless for users to efficiently implement change data capture (CDC) into their lakehouse platform with Delta Live Tables (DLT). DLT provides built-in quality controls with deep visibility into pipeline operations, observing pipeline lineage, monitoring schema, and quality checks at each step in the pipeline. DLT supports automatic error handling and best in class auto-scaling capability for streaming workloads, which enables users to have quality data with optimum resources required for their workload.

Data engineers can now easily implement CDC with a new declarative APPLY CHANGES INTO API with DLT in either SQL or Python. This new capability lets your ETL pipelines easily identify changes and apply those changes across tens of thousands of tables with low-latency support.

**Ready to get started and try out CDC in Delta Live Tables for yourself?** Please watch this webinar to learn how Delta Live Tables simplifies the complexity of data transformation and ETL, and see our Change data capture with Delta Live Tables document, official github and follow the steps in this video to create your pipeline!

SECTION 2.10

# Best Practices for Cross-Government Data Sharing

by MILOS COLIC, PRITESH PATEL, ROBERT WHIFFIN, RICHARD JAMES WILSON, MARCELL FERENCZ and EDWARD KELLY

February 21, 2023

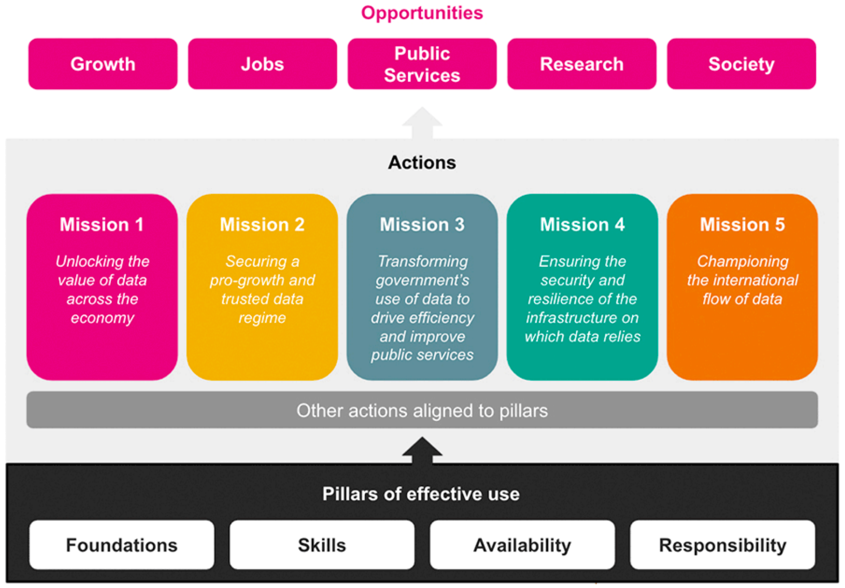
Government data exchange is the practice of sharing data between different government agencies and often partners in commercial sectors. Government can share data for various reasons, such as to improve government operations' efficiency, provide better services to the public, or support research and policy-making. In addition, data exchange in the public sector can involve sharing with the private sector or receiving data from the private sector. The considerations span multiple jurisdictions and over almost all industries. In this blog, we will address the needs disclosed as part of national data strategies and how modern technologies, particularly Delta Sharing, Unity Catalog, and clean rooms, can help you design, implement and manage a future-proof and sustainable data ecosystem.

## Data sharing and public sector

"The miracle is this: the more we share the more we have." — Leonard Nimoy.

Probably the quote about sharing that applies the most profoundly to the topic of data sharing. To the extent that the purpose of sharing the data is to create new information, new insights, and new data. The importance of data sharing is even more amplified in the government context, where federation between departments allows for increased focus. Still, the very same federation introduces challenges around data completeness, data quality, data access, security and control, FAIR-ness of data, etc. These challenges are far from trivial and require a strategic, multifaceted approach to be addressed appropriately. Technology, people, process, legal frameworks, etc., require dedicated consideration when designing a robust data sharing ecosystem.

The National Data Strategy (NDS) by the UK government outlines five actionable missions through which we can materialize the value of data for the citizen and society-wide benefits.


Opportunities
Growth
Jobs
Public Services
Research
Society
Actions
Mission 1
Unlocking the value of data across the economy
Mission 2
Securing a pro-growth and trusted data regime
Mission 3
Transforming government's use of data to drive efficiency and improve public services
Mission 4
Ensuring the security and resilience of the infrastructure on which data relies
Mission 5
Championing the international flow of data
Other actions aligned to pillars
Pillars of effective use
Foundations
Skills
Availability
Responsibility

It comes as no surprise that each and every one of the missions is strongly related to the concept of data sharing, or more broadly, data access both within and outside of government departments:

1. **Unlocking the value of the data across the economy** — Mission 1 of the NDS aims to assert government and the regulators as enablers of the value extraction from data through the adoption of best practices. The UK data economy was estimated to be near £125 billion in 2021 with an upwards trend. In this context, it is essential to understand that the government-collected and provided open data can be crucial for addressing many of the challenges across all industries.

   For example, insurance providers can better assess the risk of insuring properties by ingesting and integrating Flood areas provided by DEFRA. On the other hand, capital market investors could better understand the risk of their investments by ingesting and integrating the Inflation Rate Index by ONS. Reversely, it is crucial for regulators to have well-defined data access and data sharing patterns for conducting their regulatory activities. This clarity truly enables the economic actors that interact with government data.

2. **Securing a pro-growth and trusted data regime** — The key aspect of Mission 2 is data trust, or more broadly, adherence to data quality norms. Data quality considerations become further amplified for data sharing and data exchange use cases where we are considering the whole ecosystem at once, and quality implications transcend the boundaries of our own platform. This is precisely why we have to adopt "data sustainability." What we mean by sustainable data products are data products that harness the existing sources over reinvention of the same/similar assets, accumulation of unnecessary data (data pollutants) and that anticipate future uses.

   Ungoverned and unbounded data sharing could negatively impact data quality and hinder the growth and value of data. The quality of how the data is shared should be a key consideration of data quality frameworks. For this reason, we require a solid set of standards and best practices for data sharing with governance and quality assurance built into the process and technologies. Only this way can we ensure the sustainability of our data and secure a pro-growth trusted data regime.

3. **Transforming government's use of data to drive efficiency and improve public services** — "By 2025 data assets are organized and supported as products, regardless of whether they're used by internal teams or external customers… Data products continuously evolve in an agile manner to meet the needs of consumers… these products provide data solutions that can more easily and repeatedly be used to meet various business challenges and reduce the time and cost of delivering new AI-driven capabilities." — The data-driven enterprise of 2025 by McKinsey. AI and ML can be powerful enablers of digital transformation for both the public and private sectors.

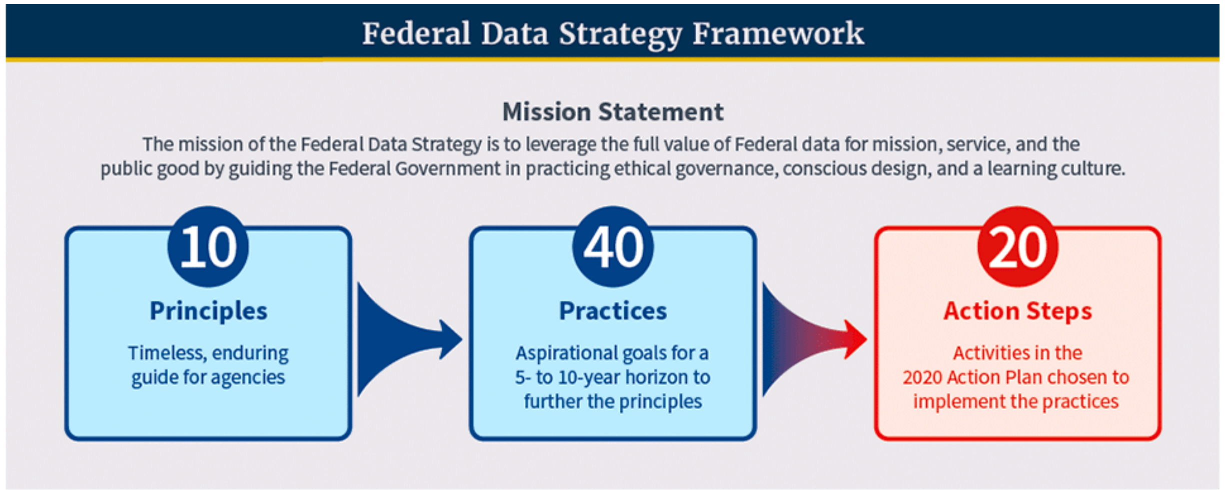
   AI, ML, reports, and dashboards are just a few examples of data products and services that extract value from data. The quality of these solutions is directly reflected in the quality of data used for building them and our ability to access and leverage available data assets both internally and externally. Whilst there is a vast amount of data available for us to build new intelligent solutions for driving efficiency for better processes, better decision-making, and better policies — there are numerous barriers that can trap the data, such as legacy systems, data silos, fragmented standards, proprietary formats, etc. Modeling data solutions as data products and standardizing them to a unified format allows us to abstract such barriers and truly leverage the data ecosystem.

4. **Ensuring the security and resilience of the infrastructure on which data relies** — Reflecting on the vision of the year 2025 — this isn't that far from now and even in a not so distant future, we will be required to rethink our approach to data, more specifically — what is our digital supply chain infrastructure/data sharing infrastructure? Data and data assets are products and should be managed as products. If data is a product, we need a coherent and unified way of providing those products.

   If data is to be used across industries and across both private and public sectors, we need an open protocol that drives adoption and habit generation. To drive adoption, the technologies we use must be resilient, robust, trusted and usable by/for all. Vendor lock-in, platform lock-in or cloud lock-in are all boundaries to achieving this vision.

5. **Championing the international flow of data** — Data exchange between jurisdictions and across governments will likely be one of the most transformative applications of data at scale. Some of the world's toughest challenges depend on the efficient exchange of data between governments — prevention of criminal activities, counterterrorism activities, net-zero emission goals, international trade, the list goes on and on. Some steps in this direction are already materializing: the U.S. federal government and UK government have agreed on data exchange for countering serious crime activities. This is a true example of championing international flow data and using data for good. It is imperative that for these use cases, we approach data sharing from a security-first angle. Data sharing standards and protocols need to adhere to security and privacy best practices.

While originally built with a focus on the UK government and how to better integrate data as a key asset of a modern government, these concepts apply in a much wider global public sector context. In the same spirit, the U.S. Federal Government proposed the Federal Data Strategy as a collection of principles, practices, action steps and timeline through which government can leverage the full value of Federal data for mission, service and the public good.



Federal Data Strategy Framework

Mission Statement

The mission of the Federal Data Strategy is to leverage the full value of Federal data for mission, service, and the public good by guiding the Federal Government in practicing ethical governance, conscious design, and a learning culture.

10 Principles — Timeless, enduring guide for agencies

40 Practices — Aspirational goals for a 5- to 10-year horizon to further the principles

20 Action Steps — Activities in the 2020 Action Plan chosen to implement the practices

The principles are grouped into three primary topics:

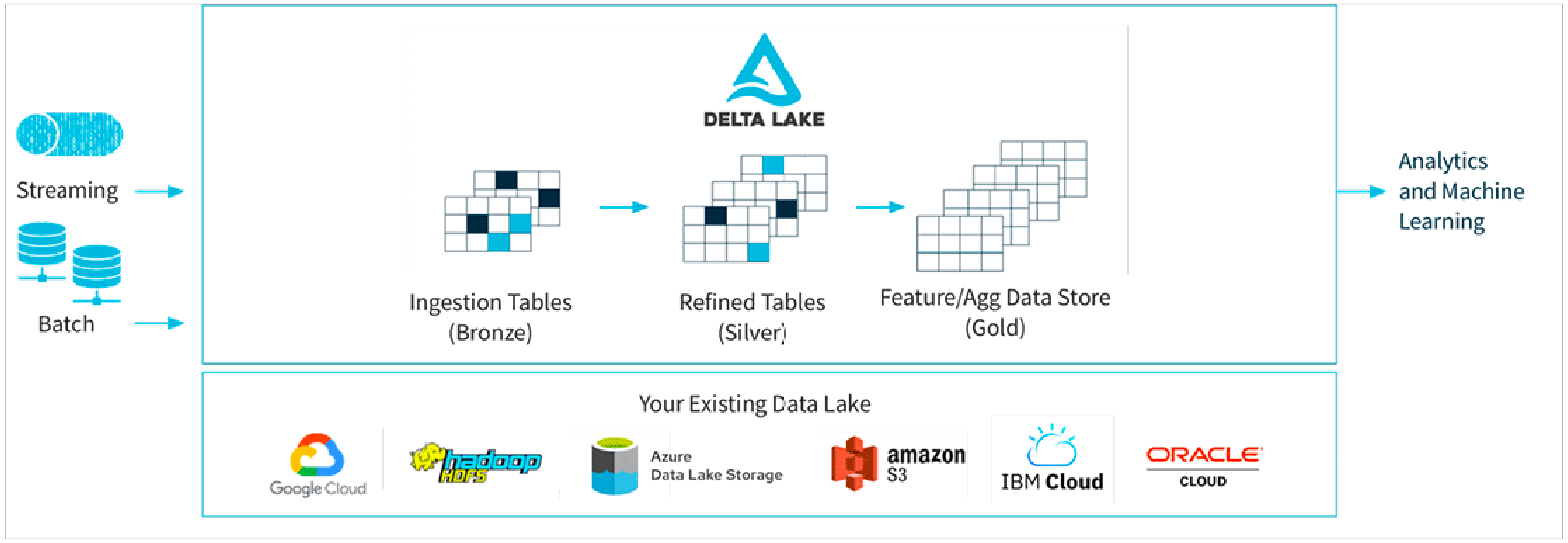
- **Ethical governance** — Within the domain of ethics, the sharing of data is a fundamental tool for promoting transparency, accountability and explainability of decision-making. It is practically impossible to uphold ethics without some form of audit conducted by an independent party. Data (and metadata) exchange is a critical enabler for continuous robust processes that ensure we are using the data for good and we are using data we can trust.
- **Conscious design** — These principles are strongly aligned with the idea of data sustainability. The guidelines promote forward thinking around usability and interoperability of the data and user-centric design principles of sustainable data products.
- **Learning culture** — Data sharing, or alternatively knowledge sharing, has an important role in building a scalable learning ecosystem and learning culture. Data is front and center of knowledge synthesis, and from a scientific angle, data proves factual knowledge. Another critical component of knowledge is the "Why?" and data is what we need to address the "Why?" component of any decisions we make, which policy to enforce, who to sanction, who to support with grants, how to improve the efficiency of government services, how to better serve citizens and society.

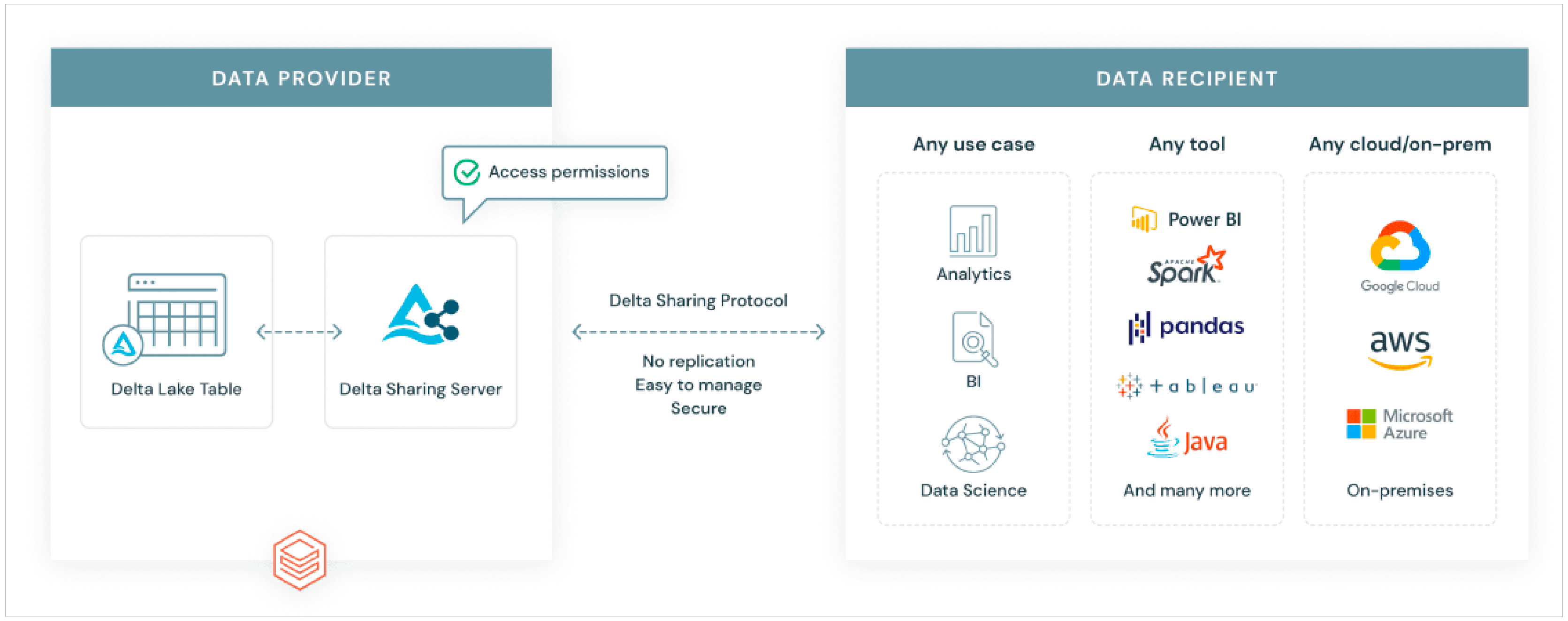
In contrast to afore discussed qualitative analysis of the value of data sharing across governments, the European Commission forecasts the economic value of the European data economy will exceed €800 billion by 2027 — roughly the same size as the Dutch economy in 2021! Furthermore, they predict more than 10 million data professionals in Europe alone. The technology and infrastructure to support the data society have to be accessible to all, interoperable, extensible, flexible and open. Imagine a world in which you'd need a different truck to transport products between different warehouses because each road requires a different set of tires — the whole supply chain would collapse. When it comes to data, we often experience the "one set of tires for one road" paradox. Rest APIs and data exchange protocols have been proposed in the past but have failed to address the need for simplicity, ease of use and cost of scaling up with the number of data products.

## Delta Sharing — the new data highway

Delta Sharing provides an open protocol for secure data sharing to any computing platform. The protocol is based on Delta data format and is agnostic concerning the cloud of choice.

Delta is an open source data format that avoids vendor, platform and cloud lock-in, thus fully adhering to the principles of data sustainability, conscious design of the U.S. Federal Data Strategy and mission 4 of the UK National Data Strategy. Delta provides a governance layer on top of the Parquet data format. Furthermore, it provides many performance optimizations not available in Parquet out of the box. The openness of the data format is a critical consideration. It is the main factor for driving the habit generation and adoption of best practices and standards.





Delta Sharing is a protocol based on a lean set of REST APIs to manage sharing, permissions and access to any data asset stored in Delta or Parquet formats. The protocol defines two main actors, the data provider (data supplier, data owner) and the data recipient (data consumer). The recipient, by definition, is agnostic to the data format at the source. Delta Sharing provides the necessary abstractions for governed data access in many different languages and tools.

Delta Sharing is uniquely positioned to answer many of the challenges of data sharing in a scalable manner within the context of highly regulated domains like the public sector:

- **Privacy and security concerns** — Personally identifiable data or otherwise sensitive or restricted data is a major part of the data exchange needs of a data-driven and modernized government. Given the sensitive nature of such data, it is paramount that the governance of data sharing is maintained in a coherent and unified manner. Any unnecessary process and technological complexities increase the risk of over-sharing data. With this in mind, Delta Sharing has been designed with security best practices from the very inception. The protocol provides end-to-end encryption, short-lived credentials, and accessible and intuitive audit and governance features. All of these capabilities are available in a centralized way across all your Delta tables across all clouds.

- **Quality and accuracy** — Another challenge of data sharing is ensuring that the data being shared is of high quality and accuracy. Given that the underlying data is stored as Delta tables, we can guarantee that the transactional nature of data is respected; Delta ensures ACID properties of data. Furthermore, Delta supports data constraints to guarantee data quality requirements at storage. Unfortunately, other formats such as CSV, CSVW, ORC, Avro, XML, etc., do not have such properties without significant additional effort. The issue becomes even more emphasized by the fact that data quality cannot be ensured in the same way on both the data provider and data recipient side without the exact reimplementation of the source systems. It is critical to embed quality and metadata together with data to ensure quality travels together with data. Any decoupled approach to managing data, metadata and quality separately increases the risk of sharing and can lead to undesirable outcomes.

- **Lack of standardization** — Another challenge of data sharing is the lack of standardization in how data is collected, organized, and stored. This is particularly pronounced in the context of governmental activities. While governments have proposed standard formats (e.g., Office for National Statistics promotes usage of CSVW), aligning all private and public sector companies to standards proposed by such initiatives is a massive challenge. Other industries may have different requirements for scalability, interoperability, format complexity, lack of structure in data, etc. Most of the currently advocated standards are lacking in multiple such aspects. Delta is the most mature candidate for assuming the central role in the standardization of data exchange format. It has been built as a transactional and scalable data format, it supports structured, semi-structured and unstructured data, it stores data schema and metadata together with data and it provides a scalable enterprise-grade sharing protocol through Delta Sharing. Finally, Delta is one of the most popular open source projects in the ecosystem and, since May 2022, has surpassed 7 million monthly downloads.

- **Cultural and organizational barriers** — These challenges can be summarized by one word: friction. Unfortunately, it's a common problem for civil servants to struggle to obtain access to both internal and external data due to over-cumbersome processes, policies and outdated standards. The principles we are using to build our data platforms and our data sharing platforms have to be self-promoting, have to drive adoption and have to generate habits that adhere to best practices.

  If there is friction with standard adoption, the only way to ensure standards are respected is by enforcement and that itself is yet another barrier to achieving data sustainability. Organizations have already adopted Delta Sharing both in the private and public sectors. For example, U.S. Citizenship and Immigration Services (USCIS) uses Delta Sharing to satisfy several interagency data-sharing requirements. Similarly, Nasdaq describes Delta Sharing as the "future of financial data sharing," and that future is open and governed.

- **Technical challenges** — Federation at the government scale or even further across multiple industries and geographies poses technical challenges. Each organization within this federation owns its platform and drives technological, architectural, platform and tooling choices.

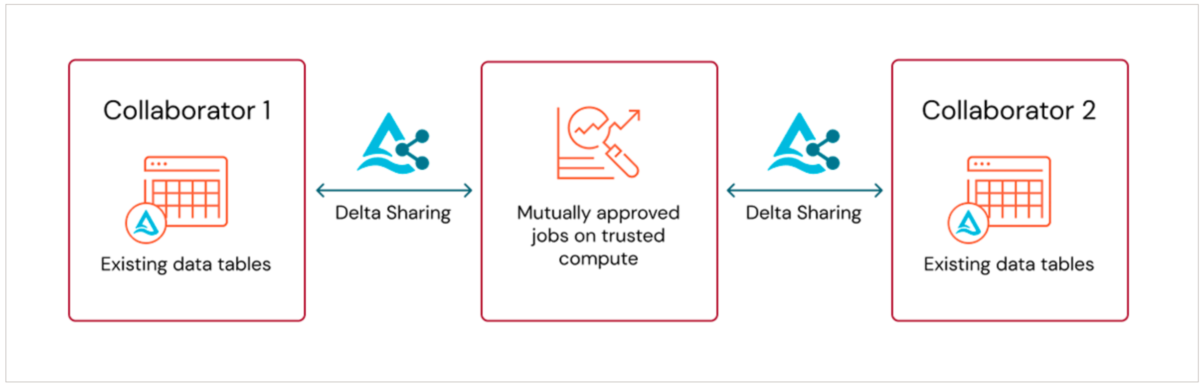
  How can we promote interoperability and data exchange in this vast, diverse technological ecosystem? The data is the only viable integration vehicle. As long as the data formats we utilize are scalable, open and governed, we can use them to abstract from individual platforms and their intrinsic complexities.

Delta format and Delta Sharing solve this wide array of requirements and challenges in a scalable, robust and open way. This positions Delta Sharing as the strongest choice for unification and simplification of the protocol and mechanism through which we share data across both private and public sectors.

## Data Sharing through data clean rooms

Taking the complexities of data sharing within highly regulated space and the public sector one step further — what if we require to share the knowledge contained in the data without ever granting direct access to the source data to external parties? These requirements may prove achievable and desirable where the data sharing risk appetite is very low.

In many public sector contexts, there are concerns that combining the data that describes citizens could lead to a big brother scenario where simply too much data about an individual is concentrated in a single data asset. If it were to fall into the wrong hands, such a hypothetical data asset could lead to immeasurable consequences for individuals and the trust in public sector services could erode. On the other hand, the value of a 360 view of the citizen could accelerate important decision-making. It could immensely improve the quality of policies and services provided to the citizens.



Data clean rooms address this particular need. With data clean rooms you can share data with third parties in a privacy-safe environment. With Unity Catalog, you can enable fine-grained access controls on the data and meet your privacy requirements. In this architecture, the data participants never get access to the raw data. The only outputs from the clean rooms are those data assets generated in a pre-agreed, governed and fully controlled manner that ensures compliance with the requirements of all parties involved.

Finally, data clean rooms and Delta Sharing can address hybrid on-premise-off-premise deployments, where the data with the most restricted access remains on the premise. In contrast, less restricted data is free to leverage the power of the cloud offerings. In said scenario, there may be a need to combine the power of the cloud with the restricted data to solve advanced use cases where capabilities are unavailable on the on-premises data platforms. Data clean rooms can ensure that no physical data copies of the raw restricted data are created, results are produced within the clean room's controlled environment and results are shared back to the on-premises environment (if the results maintain the restricted access within the defined policies) or are forwarded to any other compliant and predetermined destination system.

## Citizen value of data sharing

Every decision made by the government is a decision that affects its citizens. Whether the decision is a change to a policy, granting a benefit or preventing crime, it can significantly influence the quality of our society. Data is a key factor in making the right decisions and justifying the decisions made. Simply put, we can't expect high-quality decisions without the high quality of data and a complete view of the data (within the permitted context). Without data sharing, we will remain in a highly fragmented position where our ability to make those decisions is severely limited or even completely compromised. In this blog, we have covered several technological solutions available within the lakehouse that can derisk and accelerate how the government is leveraging the data ecosystem in a sustainable and scalable way.

For more details on the industry use cases that Delta Sharing is addressing please consult A New Approach to Data Sharing eBook.

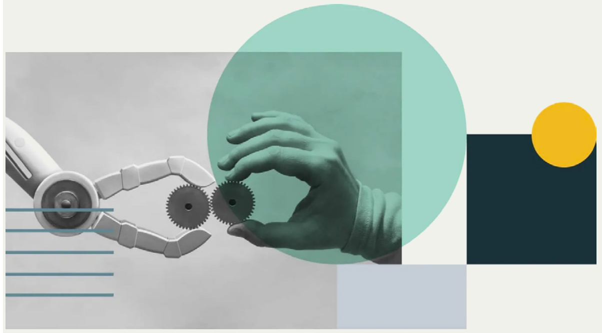
**Start experimenting with these free Databricks notebooks.**

SECTION

# 03

# Ready-to-Use Notebooks and Data Sets

This section includes several Solution Accelerators — free, ready-to-use examples of data solutions from different industries ranging from retail to manufacturing and healthcare. Each of the following scenarios includes notebooks with code and step-by-step instructions to help you get started. Get hands-on experience with the Databricks Lakehouse Platform by trying the following for yourself:



## Digital Twins

Leverage digital twins — virtual representations of devices and objects — to optimize operations and gain insights

Explore the Solution

## Overall Equipment Effectiveness

Ingest equipment sensor data for metric generation and data driven decision-making

Explore the Solution

## Recommendation Engines for Personalization

Improve customers' user experience and conversion with personalized recommendations

Explore the Solution

## Real-time point of sale analytics

Calculate current inventories for various products across multiple store locations with Delta Live Tables

Explore the Solution

## Understanding Price Transparency Data

Efficiently ingest large healthcare data sets to create price transparency for better understanding of healthcare costs

Explore the Solution

Additional Solution Accelerators with ready-to-use notebooks can be found here:

Databricks Solution Accelerators

databricks

SECTION

# 04

# Case Studies

SECTION 4.1

## Akamai delivers real-time security analytics using Delta Lake

**<1**

**Min ingestion time, reduced from 15 min**

**<85%**

**Of queries have a response time of 7 seconds or less**

INDUSTRY
Technology and Software

SOLUTION
Threat Detection

PLATFORM USE CASE
Delta Lake, Data Streaming, Photon, Databricks SQL

CLOUD
Azure

Akamai runs a pervasive, highly distributed content delivery network (CDN). Its CDN uses approximately 345,000 servers in more than 135 countries and over 1,300 networks worldwide to route internet traffic for some of the largest enterprises in media, commerce, finance, retail and many other industries. About 30% of the internet's traffic flows through Akamai servers. Akamai also provides cloud security solutions.

In 2018, the company launched a web security analytics tool that offers Akamai customers a single, unified interface for assessing a wide range of streaming security events and performing analysis of those events. The web analytics tool helps Akamai customers to take informed actions in relation to security events in real time. Akamai is able to stream massive amounts of data and meet the strict SLAs it provides to customers by leveraging Delta Lake and the Databricks Lakehouse Platform for the web analytics tool.

## Ingesting and streaming enormous amounts of data

Akamai's web security analytics tool ingests approximately 10GB of data related to security events per second. Data volume can increase significantly when retail customers conduct a large number of sales — or on big shopping days like Black Friday or Cyber Monday. The web security analytics tool stores several petabytes of data for analysis purposes. Those analyses are performed to protect Akamai's customers and provide them with the ability to explore and query security events on their own.

The web security analytics tool initially relied on an on-premises architecture running Apache Spark™ on Hadoop. Akamai offers strict service level agreements (SLAs) to its customers of 5 to 7 minutes from when an attack occurs until it is displayed in the tool. The company sought to improve ingestion and query speed to meet those SLAs. "Data needs to be as real-time as possible so customers can see what is attacking them," says Tomer Patel, Engineering Manager at Akamai. "Providing queryable data to customers quickly is critical. We wanted to move away from on-prem to improve performance and our SLAs so the latency would be seconds rather than minutes."

> **Delta Lake allows us to not only query the data better but to also acquire an increase in the data volume. We've seen an 80% increase in traffic and data in the last year, so being able to scale fast is critical.**
>
> **Tomer Patel**
> Engineering Manager, Akamai

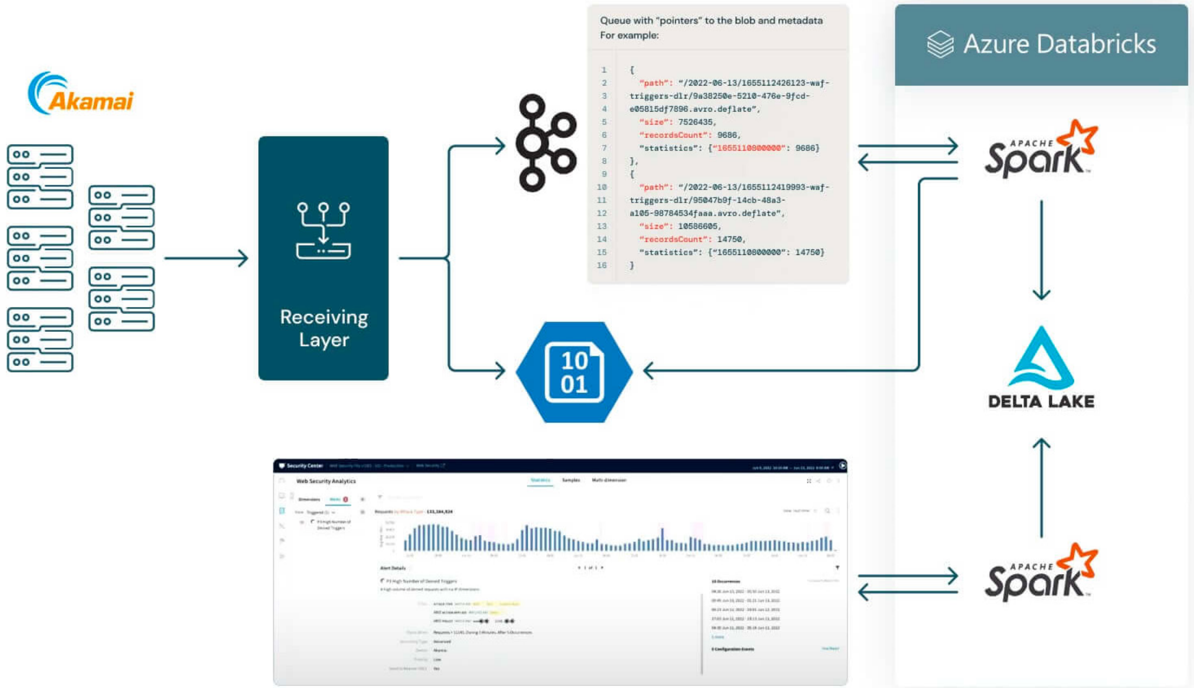
After conducting proofs of concept with several companies, Akamai chose to base its streaming analytics architecture on Spark and the Databricks Lakehouse Platform. "Because of our scale and the demands of our SLA, we determined that Databricks was the right solution for us," says Patel. "When we consider storage optimization, and data caching, if we went with another solution, we couldn't achieve the same level of performance."

## Improving speed and reducing costs

Today, the web security analytics tool ingests and transforms data, stores it in cloud storage, and sends the location of the file via Kafka. It then uses a Databricks Job as the ingest application. Delta Lake, the open source storage format at the base of the Databricks Lakehouse Platform, supports real-time querying on the web security analytics data. Delta Lake also enables Akamai to scale quickly. "Delta Lake allows us to not only query the data better but to also acquire an increase in the data volume," says Patel. "We've seen an 80% increase in traffic and data in the last year, so being able to scale fast is critical."

Akamai also uses Databricks SQL (DBSQL) and Photon, which provide extremely fast query performance. Patel added that Photon provided a significant boost to query performance. Overall, Databricks' streaming architecture combined with DBSQL and Photon enables Akamai to achieve real-time analytics, which translates to real-time business benefits.

Patel says he likes that Delta Lake is open source, as the company has benefitted from a community of users working to improve the product. “The fact that Delta Lake is open source and there’s a big community behind it means we don’t need to implement everything ourselves,” says Patel. “We benefit from fixed bugs that others have encountered and from optimizations that are contributed to the project.” Akamai worked closely with Databricks to ensure Delta Lake can meet the scale and performance requirements Akamai defined. These improvements have been contributed back to the project (many of which were made available as part of Delta Lake 2.0), and so any user running Delta Lake now benefits from the technology being tested at such a large scale in a real-world production scenario.



Queue with "pointers" to the blob and metadata
For example:

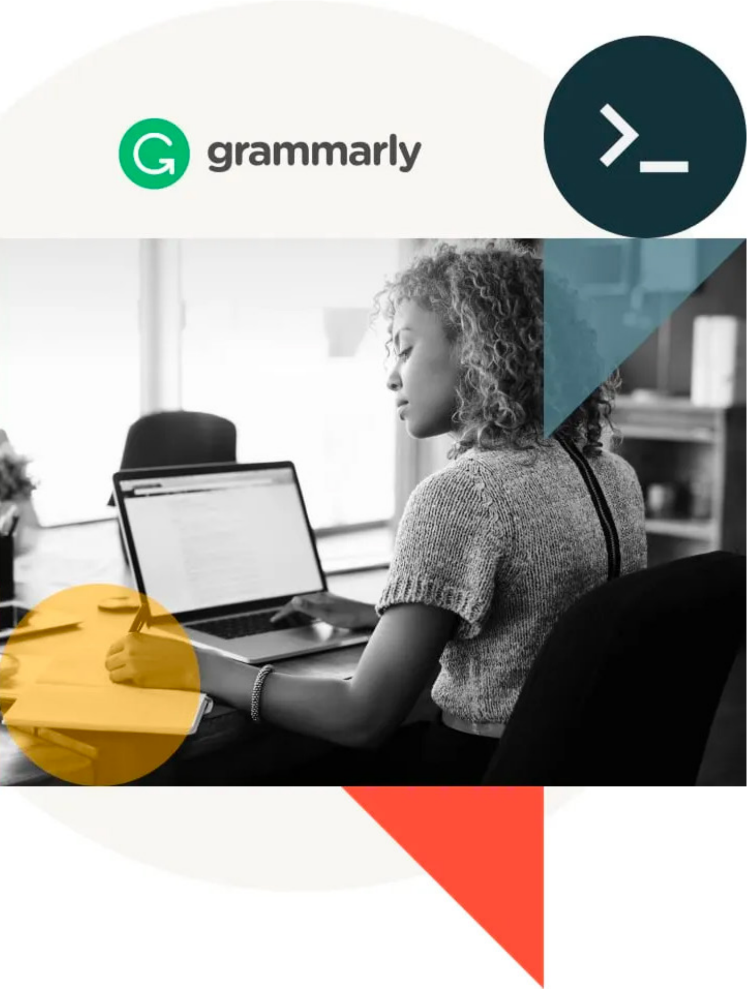
```
{
  "path": "/2022-06-13/1655112426123-waf-
triggers-dlr/9a38250e-5210-476e-9fcd-
e05815df7896.avro.deflate",
  "size": 7526435,
  "recordsCount": 9686,
  "statistics": {"1655110800000": 9686}
},
{
  "path": "/2022-06-13/1655112419993-waf-
triggers-dlr/95047b9f-14cb-48a3-
a105-98784534faaa.avro.deflate",
  "size": 10586605,
  "recordsCount": 14750,
  "statistics": {"1655110800000": 14750}
}
```

## Meeting aggressive requirements for scale, reliability and performance

Using Spark Structured Streaming on the Databricks Lakehouse Platform enables the web security analytics tool to stream vast volumes of data and provide low-latency, real-time analytics-as-a-service to Akamai’s customers. That way Akamai is able to make available security event data to customers within the SLA of 5 to 7 minutes from when an attack occurs. “Our focus is performance, performance, performance,” says Patel. “The platform’s performance and scalability are what drives us.”

Using the Databricks Lakehouse Platform, it now takes under 1 minute to ingest the security event data. “Reducing ingestion time from 15 minutes to under 1 minute is a huge improvement,” says Patel. “It benefits our customers because they can see the security event data faster and they have a view of what exactly is happening as well as the capability to filter all of it.”

Akamai’s biggest priority is to provide customers with a good experience and fast response times. To date, Akamai has moved about 70% of security event data from its on-prem architecture to Databricks, and the SLA for customer query and response time has improved significantly as a result. “Now, with the move to Databricks, our customers experience much better response time, with over 85% of queries completing under 7 seconds.” Providing that kind of real-time data means Akamai can help its customers stay vigilant and maintain an optimal security configuration.



INDUSTRY
Technology and Software

SOLUTION
Recommendation Engines, Advertising Effectiveness, Customer Lifetime Value

PLATFORM USE CASE
Lakehouse, Delta Lake, Unity Catalog, Machine Learning, ETL

CLOUD
AWS

databricks

SECTION 4.2

## Grammarly uses Databricks Lakehouse to improve user experience

**110%**
**Faster querying, at 10% of the cost to ingest, than a data warehouse**

**5 billion**
**Daily events available for analytics in under 15 minutes**

Grammarly's mission is to improve lives by improving communication. The company's trusted AI-powered communication assistance provides real-time suggestions to help individuals and teams write more confidently and achieve better results. Its comprehensive offerings — Grammarly Premium, Grammarly Business, Grammarly for Education and Grammarly for Developers — deliver leading communication support wherever writing happens. As the company grew over the years, its legacy, homegrown analytics system made it challenging to evaluate large data sets quickly and cost-effectively.

By migrating to the Databricks Lakehouse Platform, Grammarly is now able to sustain a flexible, scalable and highly secure analytics platform that helps 30 million people and 50,000 teams worldwide write more effectively every day.

## Harnessing data to improve communications for millions of users and thousands of teams

When people use Grammarly's AI communication assistance, they receive suggestions to help them improve multiple dimensions of communication, including spelling and grammar correctness, clarity and conciseness, word choice, style, and tone. Grammarly receives feedback when users accept, reject or ignore its suggestions through app-created events, which total about 5 billion events per day.

Historically, Grammarly relied on a homegrown legacy analytics platform and leveraged an in-house SQL-like language that was time-intensive to learn and made it challenging to onboard new hires. As the company grew, Grammarly data analysts found that the platform did not sufficiently meet the needs of its essential business functions, especially marketing, sales and customer success. Analysts found themselves copying and pasting data from spreadsheets because the existing system couldn't effectively ingest the external data needed to answer questions such as, "Which marketing channel delivers the highest ROI?" Reporting proved challenging because the existing system didn't support Tableau dashboards, and company leaders and analysts needed to ensure they could make decisions quickly and confidently.

> **Databricks Lakehouse has given us the flexibility to unleash our data without compromise. That flexibility has allowed us to speed up analytics to a pace we've never achieved before.**
>
> **Chris Locklin**
> Engineering Manager, Data Platforms, Grammarly

Grammarly also sought to unify its data warehouses in order to scale and improve data storage and query capabilities. As it stood, large Amazon EMR clusters ran 24/7 and drove up costs. With the various data sources, the team also needed to maintain access control. "Access control in a distributed file system is difficult, and it only gets more complicated as you ingest more data sources," says Chris Locklin, Engineering Manager, Data Platforms at Grammarly. Meanwhile, reliance on a single streaming workflow made collaboration among teams challenging. Data silos emerged as different business areas implemented analytics tools individually. "Every team decided to solve their analytics needs in the best way they saw fit," says Locklin. "That created challenges in consistency and knowing which data set was correct."

As its data strategy was evolving, Grammarly's priority was to get the most out of analytical data while keeping it secure. This was crucial because security is Grammarly's number-one priority and most important feature, both in how it protects its users' data and how it ensures its own company data remains secure. To accomplish that, Grammarly's data platform team sought to consolidate data and unify the company on a single platform. That meant sustaining a highly secure infrastructure that could scale alongside the company's growth, improving ingestion flexibility, reducing costs and fueling collaboration.

## Improving analytics, visualization and decision-making with the lakehouse

After conducting several proofs of concept to enhance its infrastructure, Grammarly migrated to the Databricks Lakehouse Platform. Bringing all the analytical data into the lakehouse created a central hub for all data producers and consumers across Grammarly, with Delta Lake at the core.

Using the lakehouse architecture, data analysts within Grammarly now have a consolidated interface for analytics, which leads to a single source of truth and confidence in the accuracy and availability of all data managed by the data platform team. Across the organization, teams are using Databricks SQL to conduct queries within the platform on both internally generated product data and external data from digital advertising platform partners. Now, they can easily connect to Tableau and create dashboards and visualizations to present to executives and key stakeholders.

"Security is of utmost importance at Grammarly, and our team's number-one objective is to own and protect our analytical data," says Locklin. "Other companies ask for your data, hold it for you, and then let you perform analytics on it. Just as Grammarly ensures our users' data always remains theirs, we wanted to ensure our company data remained ours. Grammarly's data stays inside of Grammarly."

With its data consolidated in the lakehouse, different areas of Grammarly's business can now analyze data more thoroughly and effectively. For example, Grammarly's marketing team uses advertising to attract new business. Using Databricks, the team can consolidate data from various sources to extrapolate a user's lifetime value, compare it with customer acquisition costs and get rapid feedback on campaigns. Elsewhere, data captured from user interactions flow into a set of tables used by analysts for ad hoc analysis to inform and improve the user experience.

By consolidating data onto one unified platform, Grammarly has eliminated data silos. "The ability to bring all these capabilities, data processing and analysis under the same platform using Databricks is extremely valuable," says Sergey Blanket, Head of Business Intelligence at Grammarly. "Doing everything from ETL and engineering to analytics and ML under the same umbrella removes barriers and makes it easy for everyone to work with the data and each other."

To manage access control, enable end-to-end observability and monitor data quality, Grammarly relies on the data lineage capabilities within Unity Catalog. "Data lineage allows us to effectively monitor usage of our data and ensure it upholds the standards we set as a data platform team," says Locklin. "Lineage is the last crucial piece for access control. It allows analysts to leverage data to do their jobs while adhering to all usage standards and access controls, even when recreating tables and data sets in another environment."

## Faster time to insight drives more intelligent business decisions

Using the Databricks Lakehouse Platform, Grammarly's engineering teams now have a tailored, centralized platform and a consistent data source across the company, resulting in greater speed and efficiency and reduced costs. The lakehouse architecture has led to 110% faster querying, at 10% of the cost to ingest, than a data warehouse. Grammarly can now make its 5 billion daily events available for analytics in under 15 minutes rather than 4 hours, enabling low-latency data aggregation and query optimization. This allows the team to quickly receive feedback about new features being rolled out and understand if they are being adopted as expected. Ultimately, it helps them understand how groups of users engage with the UX, improving the experience and ensuring features and product releases bring the most value to users. "Everything my team does is focused on creating a rich, personalized experience that empowers people to communicate more effectively and achieve their potential," says Locklin.

Moving to the lakehouse architecture also solved the challenge of access control over distributed file systems, while Unity Catalog enabled fine-grained, role-based access controls and real-time data lineage. "Unity Catalog gives us the ability to manage file permissions with more flexibility than a database would allow," says Locklin. "It solved a problem my team couldn't solve at scale. While using Databricks allows us to keep analytical data in-house, Unity Catalog helps us continue to uphold the highest standards of data protection by controlling access paradigms inside our data. That opens a whole new world of things that we can do."

Ultimately, migrating to the Databricks Lakehouse Platform has helped Grammarly to foster a data-driven culture where employees get fast access to analytics without having to write complex queries, all while maintaining Grammarly's enterprise-grade security practices. "Our team's mission is to help Grammarly make better, faster business decisions," adds Blanket. "My team would not be able to effectively execute on that mission if we did not have a platform like Databricks available to us." Perhaps most critically, migrating off its rigid legacy infrastructure gives Grammarly the adaptability to do more while knowing the platform will evolve as its needs evolve. "Databricks has given us the flexibility to unleash our data without compromise," says Locklin. "That flexibility has allowed us to speed up analytics to a pace we've never achieved before."

Honeywell

INDUSTRY
Manufacturing

PLATFORM USE CASE
Lakehouse, Delta Lake, Delta Live Tables

CLOUD
Azure

SECTION 4.3

## Honeywell selects Delta Live Tables for streaming data

Companies are under growing pressure to reduce energy use, while at the same time they are looking to lower costs and improve efficiency. Honeywell delivers industry-specific solutions that include aerospace products and services, control technologies for buildings and industry, and performance materials globally. Honeywell's Energy and Environmental Solutions division uses IoT sensors and other technologies to help businesses worldwide manage energy demand, reduce energy consumption and carbon emissions, optimize indoor air quality, and improve occupant well-being.

Accomplishing this requires Honeywell to collect vast amounts of data. Using Delta Live Tables on the Databricks Lakehouse Platform, Honeywell's data team can now ingest billions of rows of sensor data into Delta Lake and automatically build SQL endpoints for real-time queries and multilayer insights into data at scale — helping Honeywell improve how it manages data and extract more value from it, both for itself and for its customers.

> **Databricks helps us pull together many different data sources, do aggregations, and bring the significant amount of data we collect from our buildings under control so we can provide customers value.**
>
> **Dr. Chris Inkpen**
> Global Solutions Architect, Honeywell Energy and Environmental Solutions

databricks

## Processing billions of IoT data points per day

Honeywell's solutions and services are used in millions of buildings around the world. Helping its customers create buildings that are safe, more sustainable and productive can require thousands of sensors per building. Those sensors monitor key factors such as temperature, pressure, humidity and air quality. In addition to the data collected by sensors inside a building, data is also collected from outside, such as weather and pollution data. Another data set consists of information about the buildings themselves — such as building type, ownership, floor plan, square footage of each floor and square footage of each room. That data set is combined with the two disparate data streams, adding up to a lot of data across multiple structured and unstructured formats, including images and video streams, telemetry data, event data, etc. At peaks, Honeywell ingests anywhere between 200 to 1,000 events per second for any building, which equates to billions of data points per day. Honeywell's existing data infrastructure was challenged to meet such demand. It also made it difficult for Honeywell's data team to query and visualize its disparate data so it could provide customers with fast, high-quality information and analysis.

## ETL simplified: high-quality, reusable data pipelines

With Delta Live Tables (DLT) on the Databricks Lakehouse Platform, Honeywell's data team can now ingest billions of rows of sensor data into Delta Lake and automatically build SQL endpoints for real-time queries and multilayer insights into data at scale. "We didn't have to do anything to get DLT to scale," says Dr. Chris Inkpen, Global Solutions Architect at Honeywell Energy and Environmental Solutions. "We give the system more data, and it copes. Out of the box, it's given us the confidence that it will handle whatever we throw at it."

Honeywell credits the Databricks Lakehouse Platform for helping it to unify its vast and varied data — batch, streaming, structured and unstructured — into one platform. "We have many different data types. The Databricks Lakehouse Platform allows us to use things like Apache Kafka and Auto Loader to load and process multiple types of data and treat everything as a stream of data, which is awesome. Once we've got structured data from unstructured data, we can write standardized pipelines."

Honeywell data engineers can now build and leverage their own ETL pipelines with Delta Live Tables and gain insights and analytics quickly. ETL pipelines can be reused regardless of environment, and data can run in batches or streams. It's also helped Honeywell's data team transition from a small team to a larger team. "When we wrote our first few pipelines before DLT existed, only one person could work in one part of the functionality. Now that we've got DLT and the ability to have folders with common functionality, we've got a really good platform where we can easily spin off different pipelines."

DLT also helped Honeywell establish standard log files to monitor and cost-justify its product pipelines. "Utilizing DLT, we can analyze which parts of our pipeline need optimization," says Inkpen. "With standard pipelines, that was much more chaotic."

## Enabling ease, simplicity and scalability across the infrastructure

Delta Live Tables has helped Honeywell's data team consistently query complex data while offering simplicity of scale. It also enables end-to-end data visualization of Honeywell's data streams as they flow into its infrastructure, are transformed, and then flow out. "Ninety percent of our ETL is now captured in diagrams, so that's helped considerably and improves data governance. DLT encourages — and almost enforces — good design," says Inkpen.

Using the lakehouse as a shared workspace has helped promote teamwork and collaboration at Honeywell. "The team collaborates beautifully now, working together every day to divvy up the pipeline into their own stories and workloads," says Inkpen.

Meanwhile, the ability to manage streaming data with low latency and better throughput has improved accuracy and reduced costs. "Once we've designed something using DLT, we're pretty safe from scalability issues — certainly a hundred times better than if we hadn't written it in DLT," says Inkpen. "We can then go back and look at how we can take a traditional job and make it more performant and less costly. We're in a much better position to try and do that from DLT."

Using Databricks and DLT also helps the Honeywell team perform with greater agility, which allows them to innovate faster while empowering developers to respond to user requirements almost immediately. "Our previous architecture made it impossible to know what bottlenecks we had and what we needed to scale. Now we can do data science in near real-time."

Ultimately, Honeywell can now more quickly provide its customers with the data and analysis they need to make their buildings more efficient, healthier and safer for occupants. "I'm continuously looking for ways to improve our lifecycles, time to market, and data quality," says Inkpen. "Databricks helps us pull together many different data sources, do aggregations, and bring the significant amount of data we collect from our buildings under control so we can provide customers value."

**Ready to get started? Learn more about Delta Live Tables here.**

Wood Mackenzie
A Verisk Business

SECTION 4.4

## Wood Mackenzie helps customers transition to a more sustainable future

**12 Billion**
Data points processed each week

**80–90%**
Reduction in processing time

**Cost Savings**
In operations through workflow automation

Wood Mackenzie offers customized consulting and analysis for a wide range of clients in the energy and natural resources sectors. Founded in Edinburgh, the company first cultivated deep expertise in upstream oil and gas, then broadened its focus to deliver detailed insight for every interconnected sector of the energy, chemicals, metals and mining industries.

Today it sees itself playing an important role in the transition to a more sustainable future. Using Databricks Workflows to automate ETL pipelines helps Wood Mackenzie ingest and process massive amounts of data. Using a common workflow provided higher visibility to engineering team members, encouraging better collaboration. With an automated, transparent workflow in place, the team saw improved productivity and data quality and an easier path to fix pipeline issues when they arise.

INDUSTRY
Energy and Utilities

PLATFORM USE CASE
Lakehouse, Workflows

CLOUD
AWS

## Delivering insights to the energy industry

Fulfilling Wood Mackenzie's mission, the Lens product is a data analytics platform built to deliver insights at key decision points for customers in the energy sector. Feeding into Lens are vast amounts of data collected from various data sources and sensors used to monitor energy creation, oil and gas production, and more. Those data sources update about 12 billion data points every week that must be ingested, cleaned and processed as part of the input for the Lens platform. Yanyan Wu, Vice President of Data at Wood Mackenzie, manages a team of big data professionals that build and maintain the ETL pipeline that provides input data for Lens. The team is leveraging the Databricks Lakehouse Platform and uses Apache Spark™ for parallel processing, which provides greater performance and scalability benefits compared to an earlier single-node system working sequentially. "We saw a reduction of 80-90% in data processing time, which results in us providing our clients with more up-to-date, more complete and more accurate data," says Wu.

> **Our mission is to transform the way we power the planet. Our clients in the energy sector need data, consulting services and research to achieve that transformation. Databricks Workflows gives us the speed and flexibility to deliver the insights our clients need.**
>
> **Yanyan Wu**
> Vice President of Data, Wood Mackenzie

## Improved collaboration and transparency with a common workflow

The data pipeline managed by the team includes several stages for standardizing and cleaning raw data, which can be structured or unstructured and may be in the form of PDFs or even handwritten notes.

Different members of the data team are responsible for different parts of the pipeline, and there is a dependency between the processing stages each team member owns. Using Databricks Workflows, the team defined a common workstream that the entire team uses. Each stage of the pipeline is implemented in a Python notebook, which is run as a job in the main workflow.

Each team member can now see exactly what code is running on each stage, making it easy to find the cause of the issue. Knowing who owns the part of the pipeline that originated the problem makes fixing issues much faster. "Without a common workflow, different members of the team would run their notebooks independently, not knowing that failure in their run affected stages downstream," says Meng Zhang, Principal Data Analyst at Wood Mackenzie. "When trying to rerun notebooks, it was hard to tell which notebook version was initially run and the latest version to use."

Using Workflows' alerting capabilities to notify the team when a workflow task fails ensures everyone knows a failure occurred and allows the team to work together to resolve the issue quickly. The definition of a common workflow created consistency and transparency that made collaboration easier. "Using Databricks Workflows allowed us to encourage collaboration and break up the walls between different stages of the process," explains Wu. "It allowed us all to speak the same language."

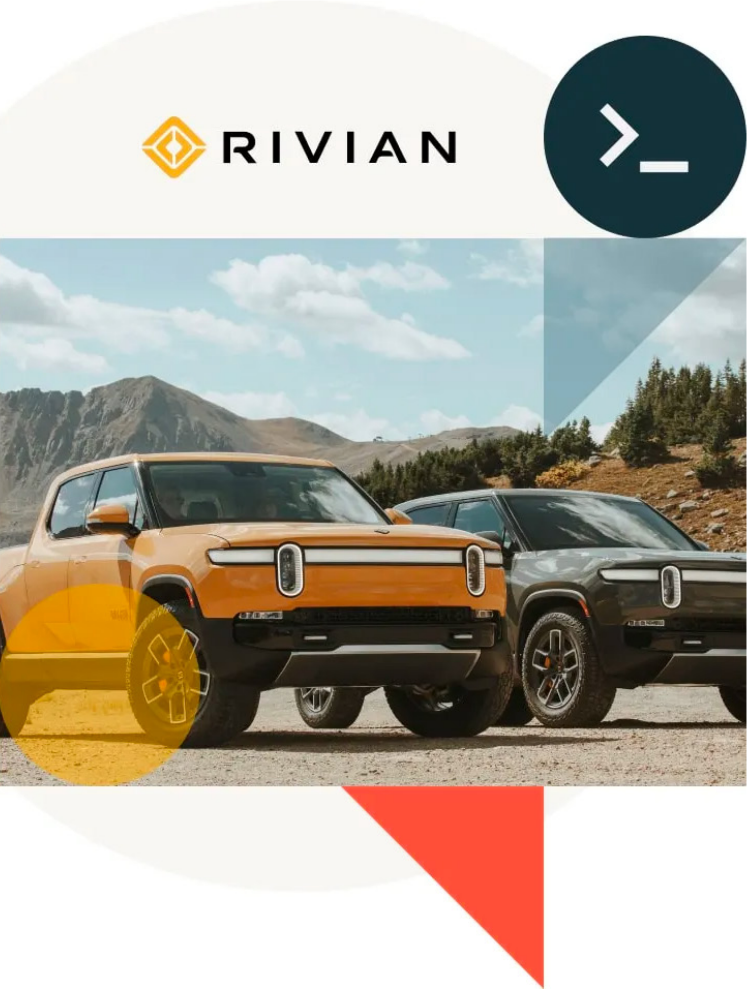
Creating transparency and consistency is not the only advantage the team saw. Using Workflows to automate notebook runs also led to cost savings compared to running interactive notebooks manually.

## Improved code development productivity

The team's ETL pipeline development process involves iteration on PySpark notebooks. Leveraging interactive notebooks in the Databricks UI makes it easy for data professionals on the team to manually develop and test a notebook. Because Databricks Workflows supports running notebooks as task type (along with Python files, JAR files and other types), when the code is ready for production, it's easy and cost effective to automate it by adding it to a workflow. The workflow can then be easily revised by adding or removing any steps to or from the defined flow. This way of working keeps the benefit of manually developing notebooks with the interactive notebook UI while leveraging the power of automation, which reduces potential issues that may happen when running notebooks manually.

The team has gone even further in increasing productivity by developing a CI/CD process. "By connecting our source control code repository, we know the workflow always runs the latest code version we committed to the repo," explains Zhang. "It's also easy to switch to a development branch to develop a new feature, fix a bug and run a development workflow. When the code passes all tests, it is merged back to the main branch and the production workflow is automatically updated with the latest code."

Going forward, Wood Mackenzie plans to optimize its use of Databricks Workflows to automate machine learning processes such as model training, model monitoring and handling model drift. The firm uses ML to improve its data quality and extract insights to provide more value to its clients. "Our mission is to transform how we power the planet," Wu says. "Our clients in the energy sector need data, consulting services and research to achieve that transformation. Databricks Workflows gives us the speed and flexibility to deliver the insights our clients need."



SECTION 4.5

## Rivian redefines driving experience with the Databricks Lakehouse

### 250 platform users

**A 50x increase from a year ago**

Rivian is preserving the natural world for future generations with revolutionary Electric Adventure Vehicles (EAVs). With over 25,000 EAVs on the road generating multiple terabytes of IoT data per day, the company is using data insights and machine learning to improve vehicle health and performance. However, with legacy cloud tooling, it struggled to scale pipelines cost-effectively and spent significant resources on maintenance — slowing its ability to be truly data driven.

Since moving to the Databricks Lakehouse Platform, Rivian can now understand how a vehicle is performing and how this impacts the driver using it. Equipped with these insights, Rivian is innovating faster, reducing costs, and ultimately, delivering a better driving experience to customers.

INDUSTRY
Manufacturing

SOLUTION
Predictive Maintenance, Scaling ML Models for IoT, Data-Driven ESG

PLATFORM
Lakehouse, Delta Lake, Unity Catalog

CLOUD
AWS

## Struggling to democratize data on a legacy platform

Building a world that will continue to be enjoyed by future generations requires a shift in the way we operate. At the forefront of this movement is Rivian — an electric vehicle manufacturer focused on shifting our planet's energy and transportation systems entirely away from fossil fuel. Today, Rivian's fleet includes personal vehicles and involves a partnership with Amazon to deliver 100,000 commercial vans. Each vehicle uses IoT sensors and cameras to capture petabytes of data ranging from how the vehicle drives to how various parts function. With all this data at its fingertips, Rivian is using machine learning to improve the overall customer experience with predictive maintenance so that potential issues are addressed before they impact the driver.

Before Rivian even shipped its first EAV, it was already up against data visibility and tooling limitations that decreased output, prevented collaboration and increased operational costs. It had 30 to 50 large and operationally complicated compute clusters at any given time, which was costly. Not only was the system difficult to manage, but the company experienced frequent cluster outages as well, forcing teams to dedicate more time to troubleshooting than to data analysis. Additionally, data silos created by disjointed systems slowed the sharing of data, which further contributed to productivity issues. Required data languages and specific expertise of toolsets created a barrier to entry that limited developers from making full use of the data available. Jason Shiverick, Principal Data Scientist at Rivian, said the biggest issue was the data access. "I wanted to open our data to a broader audience of less technical users so they could also leverage data more easily."

Rivian knew that once its EAVs hit the market, the amount of data ingested would explode. In order to deliver the reliability and performance it promised, Rivian needed an architecture that would not only democratize data access, but also provide a common platform to build innovative solutions that can help ensure a reliable and enjoyable driving experience.

> **Databricks Lakehouse empowers us to lower the barrier of entry for data access across our organization so we can build the most innovative and reliable electric vehicles in the world.**
>
> **Wassym Bensaid**
> Vice President of Software Development, Rivian

## Predicting maintenance issues with Databricks Lakehouse

Rivian chose to modernize its data infrastructure on the Databricks Lakehouse Platform, giving it the ability to unify all of its data into a common view for downstream analytics and machine learning. Now, unique data teams have a range of accessible tools to deliver actionable insights for different use cases, from predictive maintenance to smarter product development. Venkat Sivasubramanian, Senior Director of Big Data at Rivian, says, "We were able to build a culture around an open data platform that provided a system for really democratizing data and analysis in an efficient way." Databricks' flexible support of all programming languages and seamless integration with a variety of toolsets eliminated access roadblocks and unlocked new opportunities. Wassym Bensaid, Vice President of Software Development at Rivian, explains, "Today we have various teams, both technical and business, using Databricks Lakehouse to explore our data, build performant data pipelines, and extract actionable business and product insights via visual dashboards."

Rivian's ADAS (advanced driver-assistance systems) Team can now easily prepare telemetric accelerometer data to understand all EAV motions. This core recording data includes information about pitch, roll, speed, suspension and airbag activity, to help Rivian understand vehicle performance, driving patterns and connected car system predictability. Based on these key performance metrics, Rivian can improve the accuracy of smart features and the control that drivers have over them. Designed to take the stress out of long drives and driving in heavy traffic, features like adaptive cruise control, lane change assist, automatic emergency driving, and forward collision warning can be honed over time to continuously optimize the driving experience for customers.

Secure data sharing and collaboration was also facilitated with the Databricks Unity Catalog. Shiverick describes how unified governance for the lakehouse benefits Rivian productivity. "Unity Catalog gives us a truly centralized data catalog across all of our different teams," he said. "Now we have proper access management and controls." Venkat adds, "With Unity Catalog, we are centralizing data catalog and access management across various teams and workspaces, which has simplified governance." End-to-end version controlled governance and auditability of sensitive data sources, like the ones used for autonomous driving systems, produces a simple but secure solution for feature engineering. This gives Rivian a competitive advantage in the race to capture the autonomous driving grid.

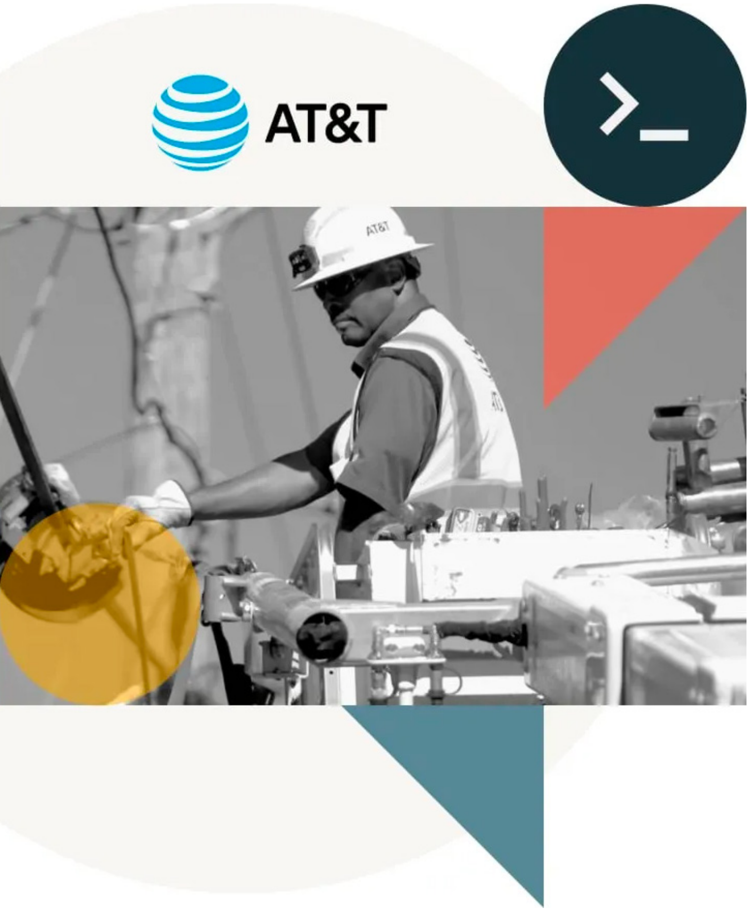
## Accelerating into an electrified and sustainable world

By scaling its capacity to deliver valuable data insights with speed, efficiency and cost-effectiveness, Rivian is primed to leverage more data to improve operations and the performance of its vehicles to enhance the customer experience. Venkat says, "The flexibility that lakehouse offers saves us a lot of money from a cloud perspective, and that's a huge win for us." With Databricks Lakehouse providing a unified and open source approach to data and analytics, the Vehicle Reliability Team is able to better understand how people are using their vehicles, and that helps to inform the design of future generations of vehicles. By leveraging the Databricks Lakehouse Platform, they have seen a 30%–50% increase in runtime performance, which has led to faster insights and model performance.

Shiverick explains, "From a reliability standpoint, we can make sure that components will withstand appropriate lifecycles. It can be as simple as making sure door handles are beefy enough to endure constant usage, or as complicated as predictive and preventative maintenance to eliminate the chance of failure in the field. Generally speaking, we're improving software quality based on key vehicle metrics for a better customer experience."

From a design optimization perspective, Rivian's unobstructed data view is also producing new diagnostic insights that can improve fleet health, safety, stability and security. Venkat says, "We can perform remote diagnostics to triage a problem quickly, or have a mobile service come in, or potentially send an OTA to fix the problem with the software. All of this needs so much visibility into the data, and that's been possible with our partnership and integration on the platform itself." With developers actively building vehicle software to improve issues along the way.

Moving forward, Rivian is seeing rapid adoption of Databricks Lakehouse across different teams — increasing the number of platform users from 5 to 250 in only one year. This has unlocked new use cases including using machine learning to optimize battery efficiency in colder temperatures, increasing the accuracy of autonomous driving systems, and serving commercial depots with vehicle health dashboards for early and ongoing maintenance. As more EAVs ship, and its fleet of commercial vans expands, Rivian will continue to leverage the troves of data generated by its EAVs to deliver new innovations and driving experiences that revolutionize sustainable transportation.



INDUSTRY
Communication Service Providers

SOLUTION
Customer Retention, Subscriber Churn Prediction, Threat Detection

PLATFORM
Lakehouse, Data Science, Machine Learning, Data Streaming

CLOUD
Azure

SECTION 4.6

## Migrating to the cloud to better serve millions of customers

**300%**
ROI from OpEx savings and cost avoidance

**3X**
Faster delivery of ML/data science use cases

Consistency in innovation is what keeps customers with a telecommunications company and is why AT&T is ranked among the best. However, AT&T's massive on-premises legacy Hadoop system proved complex and costly to manage, impeding operational agility and efficiency and engineering resources. The need to pivot to cloud to better support hundreds of millions of subscribers was apparent.

Migrating from Hadoop to Databricks on the Azure cloud, AT&T experienced significant savings in operating costs. Additionally, the new cloud-based environment has unlocked access to petabytes of data for correlative analytics and an AI-as-a-Service offering for 2,500+ users across 60+ business units. AT&T can now leverage all its data — without overburdening its engineering team or exploding operational costs — to deliver new features and innovations to its millions of end users.

## Hadoop technology adds operational complexity and unnecessary costs

AT&T is a technology giant with hundreds of millions of subscribers and ingests 10+ petabytes[a] of data across the entire data platform each day. To harness this data, it has a team of 2,500+ data users across 60+ business units to ensure the business is data powered — from building analytics to ensure decisions are based on the best data-driven situation awareness to building ML models that bring new innovations to its customers. To support these requirements, AT&T needed to democratize and establish a data single version of truth (SVOT) while simplifying infrastructure management to increase agility and lower overall costs.

However, physical infrastructure was too resource intensive. The combination of a highly complex hardware setup (12,500 data sources and 1,500+ servers) coupled with an on-premises Hadoop architecture proved complex to maintain and expensive to manage. Not only were the operational costs to support workloads high, but there were also additional capital costs around data centers, licensing and more. Up to 70% of the on-prem platform had to be prioritized to ensure 50K data pipeline jobs succeeded and met SLAs and data quality objectives. Engineers' time was focused on managing updates, fixing performance issues or simply provisioning resources rather than focusing on higher-valued tasks. The resource constraints of physical infrastructure also drove serialization of data science activities, slowing innovation. Another hurdle faced in operationalizing petabytes of data was the challenge of building streaming data pipelines for real-time analytics, an area that was key to supporting innovative use cases required to better serve its customers.

With these deeply rooted technology issues, AT&T was not in the best position to achieve its goals of increasing its use of insights for improving its customer experience and operating more efficiently. "To truly democratize data across the business, we needed to pivot to a cloud-native technology environment," said Mark Holcomb, Distinguished Solution Architect at AT&T. "This has freed up resources that had been focused on managing our infrastructure and move them up the value chain, as well as freeing up capital for investing in growth-oriented initiatives."

## A seamless migration journey to Databricks

As part of its due diligence, AT&T ran a comprehensive cost analysis and concluded that Databricks was both the fastest and achieved the best price/performance for data pipelines and machine learning workloads. AT&T knew the migration would be a massive undertaking. As such, the team did a lot of upfront planning — they prioritized migrating their largest workloads first to immediately reduce their infrastructure footprint. They also decided to migrate their data before migrating users to ensure a smooth transition and experience for their thousands of data practitioners.

> **The migration from Hadoop to Databricks enables us to bring more value to our customers and do it more cost-efficiently and much faster than before.**
>
> **Mark Holcomb**
> Distinguished Solution Architect, AT&T

They spent a year deduplicating and synchronizing data to the cloud before migrating any users. This was a critical step in ensuring the successful migration of such a large, complex multi-tenant environment of 2,500+ users from 60+ business units and their workloads. The user migration process occurred over nine months and enabled AT&T to retire on-premises hardware in parallel with migration to accelerate savings as early as possible. Plus, due to the horizontal, scalable nature of Databricks, AT&T didn't need to have everything in one contiguous environment. Separating data and compute, and across multiple accounts and workspaces, ensured analytics worked seamlessly without any API call limits or bandwidth issues and consumption clearly attributed to the 60+ business units.

All in all, AT&T migrated over 1,500 servers, more than 50,000 production CPUs, 12,500 data sources and 300 schemas. The entire process took about two and a half years. And it was able to manage the entire migration with the equivalent of 15 full-time internal resources. "Databricks was a valuable collaborator throughout the process," said Holcomb. "The team worked closely with us to resolve product features and security concerns to support our migration timeline."

## Databricks reduces TCO and opens new paths to innovation

One of the immediate benefits of moving to Databricks was huge cost savings. AT&T was able to rationalize about 30% of its data by identifying and not migrating underutilized and duplicate data. And prioritizing the migration of the largest workloads allowed half the on-prem equipment to be rationalized during the course of the migration. "By prioritizing the migration of our most compute-intensive workloads to Databricks, we were able to significantly drive down costs while putting us in position to scale more efficiently moving forward," explained Holcomb. The result is an anticipated 300% five-year migration ROI from OpEx savings and cost avoidance (e.g., not needing to refresh data center hardware).

With data readily available and the means to analyze data at any scale, teams of citizen data scientists and analysts can now spend more time innovating, instead of serializing analytics efforts or waiting on engineering to provide the necessary resources — or having data scientists spend their valuable time on less complex or less insightful analyses. Data scientists are now able to collaborate more effectively and speed up machine learning workflows so that teams can deliver value more quickly, with a 3x faster time to delivery for new data science use cases.

"Historically you would have had operations in one system and analytics in a separate one," said Holcomb. "Now we can do more use cases like operational analytics in a platform that fosters cross-team collaboration, reduces cost and improves the consistency of answers." Since migrating to Databricks, AT&T now has a single version of truth to create new data-driven opportunities, including a self-serve AI-as-a-Service analytics platform that will enable new revenue streams and help it continue delivering exceptional innovations to its millions of customers.

# About Databricks

Databricks is the data and AI company. More than 9,000 organizations worldwide — including Comcast, Condé Nast and over 50% 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.

START YOUR FREE TRIAL

Contact us for a personalized demo
**databricks.com/contact**

databricks