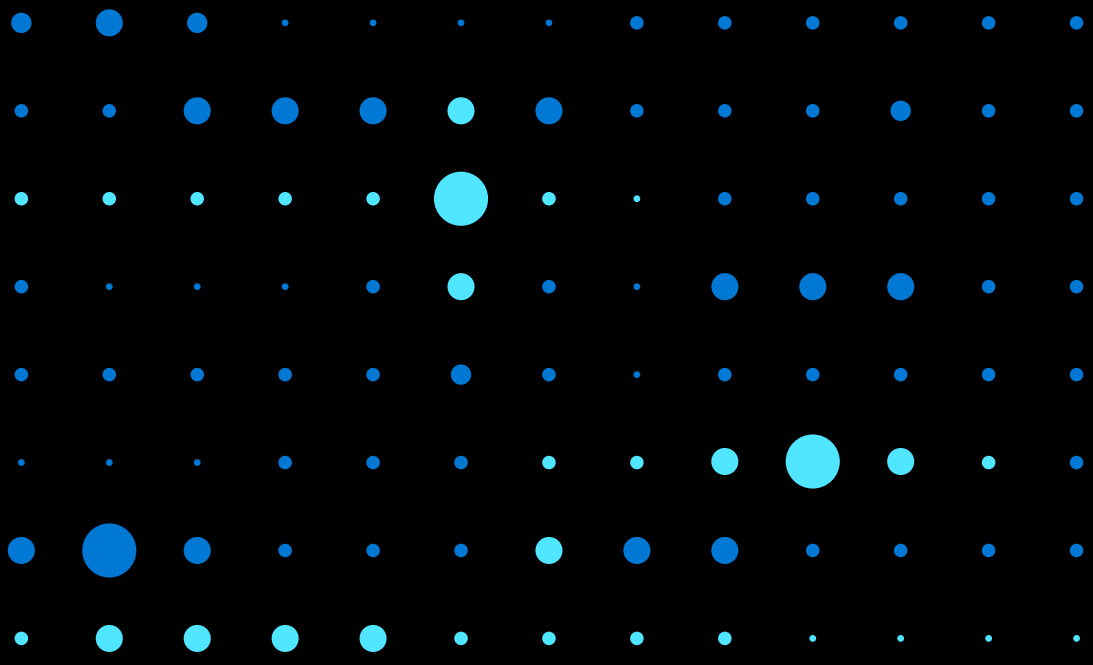
Azure Databricks

Delta Lake on Azure Databricks



Microsoft Azure

Contents

An “under the hood” look

Delta Lake on Azure Databricks allows you to configure Delta Lake based on your workload patterns and has optimized layouts and indexes for fast interactive queries. Delta Lake is an open source storage layer that brings reliability to data lakes. It provides ACID transactions, scalable metadata handling, and unifies streaming and batch data processing. Delta Lake runs on top of your existing data lake and is fully compatible with Apache Spark APIs.

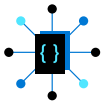
Designed for both batch and stream processing, Delta Lake also addresses concerns about system complexity. Its advanced architecture enables high reliability and low latency, using techniques such as schema validation, compaction, data skipping, and more to address pipeline development, data management and as well as query serving.

Challenges in Harnessing Data

To respond to ever-growing data volumes, many organizations are collecting their data in data lakes ahead of making it available for analysis. This has addressed some of the challenges of harnessing structured and unstructured data, but, unfortunately, data lakes suffer from some key challenges of their own:

Query performance
The ETL processes required by data lakes can add significant latency to the point that it may take hours before incoming data manifests in a query response. As a result, the users do not benefit from the latest data. Further, the longer query run times can be deemed unacceptable by users, especially when the scale increases.

Data reliability
The complex data pipelines are error-prone and complicated, consuming resources significantly. Further, schema evolution as business needs change can be effort intensive. Finally, errors or gaps in incoming data, a common occurrence, can cause failures in downstream applications.



System complexity
It is difficult to build flexible data engineering pipelines that combine streaming and batch analytics. These systems require complex and low-level code. Intervention during stream processing with batch correction or programming multiple streams from the same sources or to the same destinations is restricted.

Practitioners typically organize their pipelines using a multi-hop architecture. The pipeline starts with a "firehose" of records from many different parts of the organization. These data are then normalized and enriched with dimension information.

Following this, the data may be filtered down and aggregated for particular business objectives. Finally, high-level summaries of key business metrics might be created.

There are various challenges encountered during the pipeline stages:

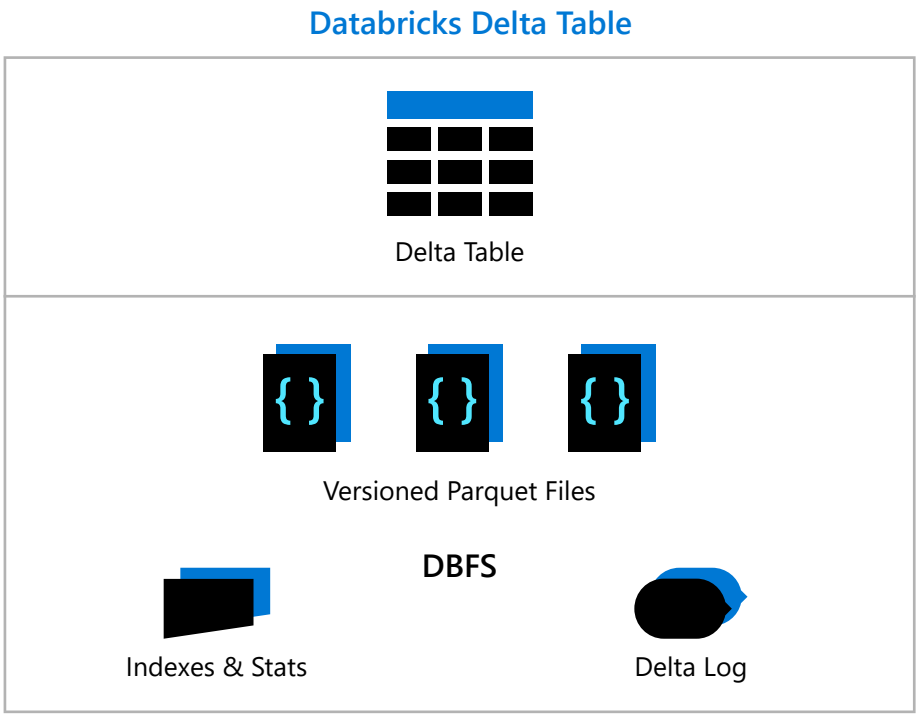
- Schema changes can break enrichment, joins, and transforms between stages.
- Failures may cause data between stages to either drop on the floor or be duplicated.
- Partitioning alone does not scale for multidimensional data.
- Standard tables do not allow the combination of streaming and batch processes for optimal latencies.
- Concurrent access suffers from inconsistent query results.
- Failing streaming jobs can require resetting and restarting data processing.

Delta Lake on Azure Databricks addresses the challenges faced by data engineering professionals in marshalling their data head-on. It offers a simple analytics architecture that can address both batch and stream use cases with high query performance and high data reliability.

Delta Lake Architecture

The Delta Lake architecture provides an efficient and transactional way to work with large data sets stored as files on Azure Blob. It employs an ordered log (the Delta Lake transaction log) of atomic collections of actions (e.g. AddFile, RemoveFile, etc.). Delta Lake tables are build built atop the Azure Databricks File System (DBFS). which manifests:

- Versioned Parquet files (based on Apache Parquet[1])
- Indexes and stats
- The Delta Lake transaction log



Parquet is a columnar storage format and supports efficient compression and encoding schemes. In Delta Lake, the versioned Parquet files enable you to track the evolution of the data. Indexes and statistics about the files are maintained to increase query efficiency. The Delta Lake transaction log can be appended to by multiple writers that are mediated by optimistic

1 https://parquet.apache.org/

concurrency control that provide serializable ACID transactions. Changes to the table are stored as ordered atomic units called commits. The log can be read in parallel by a cluster of Spark executors.

The Delta Lake on Azure Databricks design allows readers to efficiently query a snapshot of the state of a table, optionally filtering by partition value. The result is fast operations irrespective of the number of files.

Building and Maintaining Robust Pipelines

Designing and building robust pipelines is the first step in realizing value from one's data resources. The focus should be on:

- Ingesting the data
- Ensuring data quality upon ingest
- Maintaining the data over time as enhancements and corrections need to be made
- Managing the environment effectively as queries as being run.

Delta Lake on Azure Databricks helps address challenges throughout the various pipeline stages and handles both batch and streaming data.

Delta Lake on Azure Databricks Details

Delta Lake uses a number of techniques to address query performance, data reliability and system complexity.

Query performance

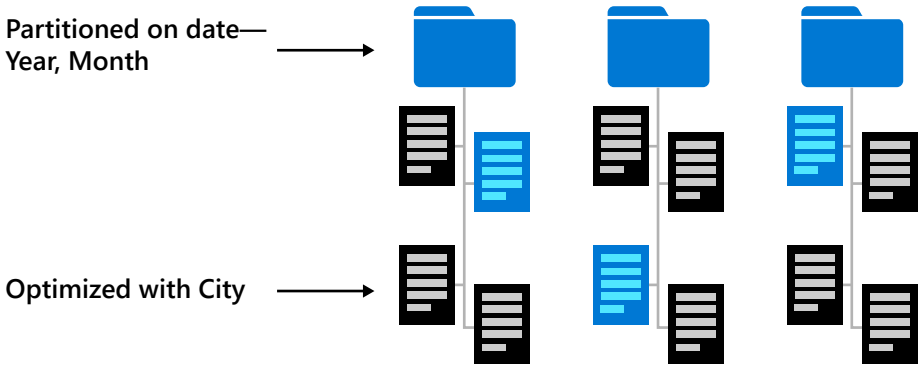
Query performance is a major driver of user satisfaction. The faster a query returns results, the sooner those results are available for using. With Delta Lake on Azure Databricks, query performance can be 10 to 100 times faster than with Apache Spark on Parquet. Delta Lake on Azure Databricks employs various techniques to deliver superior performance.

Data indexing

Delta Lake creates and maintains indexes of the ingested data, which increases the querying speed significantly.

Data skipping

Delta Lake maintains file statistics so that data subsets relevant to the query are used instead of entire tables—this partition pruning avoids processing data that is not relevant to the query. Multidimensional clustering (using Z-ordering algorithm) is used to enable this. This technique is particularly helpful in the case of complex queries.




```
SELECT ... FROM flights
WHERE date BETWEEN "2020-Jun-01" AND "2020-Aug-31"
      AND originating_city = "Seattle"
```

Only read the relevant subsets (shown in blue) to fulfil query

Compaction

Often, especially in the case of streaming data, a large number of small files are created as data is ingested. Storing and accessing these small files can be processing-intensive, slow and inefficient from a storage utilization perspective. Delta Lake on Azure Databricks manages file sizes (i.e., compacts or combines multiple small files into more efficient larger ones) to speed up query performance.

Data Caching

Accessing data from storage repeatedly can slow query performance. Delta Lake automatically caches highly accessed data to speed access for queries.

Data Reliability

Reliable datasets are key to successful data analytics, whether it's feeding dashboards or enabling ML initiatives. Delta uses various techniques to achieve data reliability. By 'filtering' messy data and blocking access into Delta Lake, clean data sits in a Delta Lake on top of the data lake. Optimistic concurrency control between writes and snapshot isolation deliver ACID transactions for consistent reads during writes. Delta Lake also offers built-in data versioning for easy rollbacks and reproducing reports.

ACID Transactions

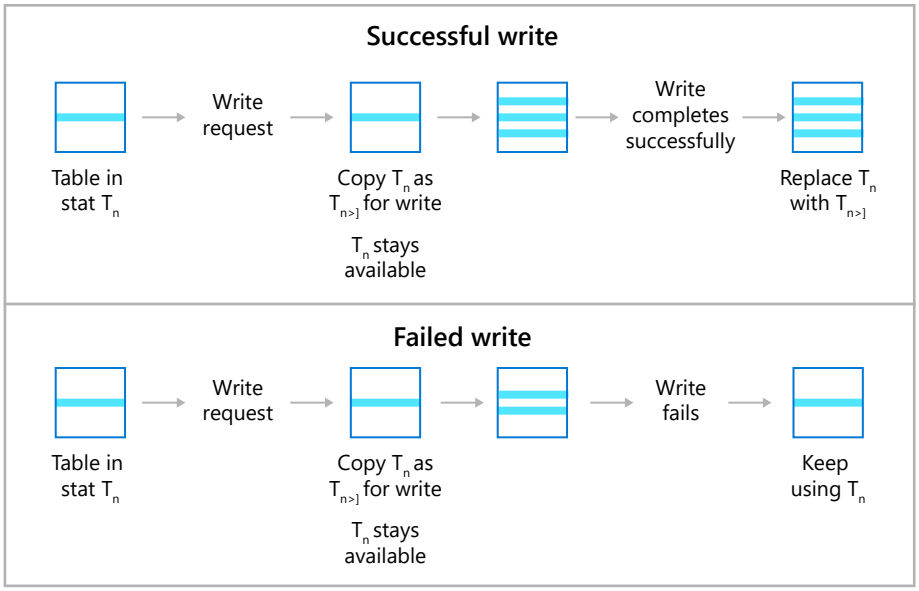
The inevitable partial or failed writes risk corrupting the data. Delta Lake employs an "all or nothing" ACID transaction approach to prevent such corruption.

Snapshot Isolation

In large environments with multiple concurrent readers and writers, metadata must be maintained so that the reads in progress act on consistent views of data and are not impacted by writes in progress. Delta Lake on Azure Databricks provides snapshot isolation, ensuring that multiple writers can write to a dataset simultaneously without interfering with jobs reading the dataset.

Schema Enforcement

Notionally similar data but from different sources or of different vintage can differ in its representation, creating difficulties for using it effectively. Delta Lake helps ensure data integrity for ingested data by providing schema enforcement. Data can be stored using the preferred schema and potential data corruption with incorrect or invalid schemas is avoided.



Exactly Once

When working with long-running computations, multiple streams or concurrent batch jobs, there is a risk that some data will be missed (due to transmission difficulties) or duplicated (in attempts to correct for the misses). Delta Lake employs checkpointing to provide a robust exactly once delivery semantic that ensures that data is neither missed nor repeated erroneously.

UPSERTS and DELETES Support

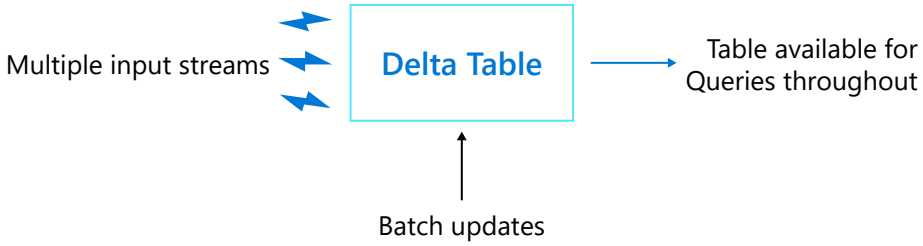
Standard Spark tables are write-once, i.e., they cannot be modified. Any necessary changes to account for late-arriving data or data that requires updating must be addressed using new tables. Delta Lake on Azure Databricks provides support for UPSERTS and DELETES. These commands provide a more "convenient" way of dealing with such changes.

System Complexity

System complexity is a key determinant not only of reliability and cost-effectiveness but also, very importantly, of responsiveness. As business requirements evolve, data analytics architecture needs to be flexible and responsive to keep up.

Unified Batch/Stream

Delta Lake on Azure Databricks handles both batch and streaming data (via a direct integration with Structured Streaming for low latency updates). It can also concurrently write batch and streaming data to the same data table. Not only does this result in a simpler system architecture, but it also results in shorter time from data ingest to query result.



Schema Evolution

Delta Lake on Azure Databricks can infer schema from input data. This reduces the effort for dealing with schema impact of changing business needs at multiple levels of the pipeline/data stack.

Delta Lake on Azure Databricks Best Practices

Adopting the following best practices will help you make the most of Delta Lake on Azure Databricks.

Go Through Delta Lake

All writes and reads should go through Delta Lake to ensure consistent overall behavior. Further, Delta Lake tables must not be accessed using earlier versions of the Azure Databricks Runtime because those versions do not understand Delta Lake and do not support it.

Run OPTIMIZE Regularly

The OPTIMIZE command triggers compaction. It makes no data related changes to the table, so a read before and after an OPTIMIZE has the same results. It should be run regularly on tables that analysts are querying to ensure efficiency. A good starting point is to do this every day. Note that OPTIMIZE should not be run on base or staging tables.

Run VACUUM Regularly

To ensure that concurrent readers can continue reading a stale snapshot of a table, Delta Lake on Azure Databricks leaves deleted files on DBFS for a period of time. The VACUUM command helps save on storage costs by cleaning up these invalid files. It can, however, interrupt users querying a Delta table similar to when partitions are re-written. VACUUM should be run regularly to clean up expired snapshots that are no longer required.

Batch Modifications

The Parquet files that underpin Delta Lake are immutable and thus need to be rewritten completely to reflect changes regardless of the extent of the change. Use MERGE INTO to batch changes to amortize costs.

Use DELETEs

Manually deleting files from the underlying storage is likely to break the Delta Lake table. Instead, you should use DELETE commands to ensure proper progression of the change.

Trying Delta Lake on Azure Databricks

Delta Lake is easy to put into production—Databricks on Azure users have been able to get into production with a Delta Lake-based solution in just a few weeks, using a small team compared to alternate approaches that take much longer and more resources. It is easy to get started with Delta Lake.

Porting existing Spark code for using Delta Lake is as simple as changing
“CREATE TABLE ... USING parquet” to
“CREATE TABLE ... USING delta”

or changing
“dataframe.write.format(“parquet”).load(“/data/events”)”
“dataframe.write.format(“delta”).load(“/data/events”)”

If you are already using Azure Databricks, you can explore Delta Lake today using:

- The Delta Lake on Azure Databricks quickstart
- Optimization notebooks

You can learn more about Delta Lake on Azure Databricks from the documentation.

Resources

Documentation

- Azure Databricks best practices
- Introduction to Delta Lake
- Introduction to MLFlow

Webinars

- Get started with Apache Spark
- Azure Databricks best practices
- Machine Learning life cycle management with Azure Databricks and Azure Machine Learning

More info

- Azure Databricks pricing page
- Azure Databricks unit pre-purchase plan

Contact us

- Talk to a sales expert

Invent with purpose

Microsoft Azure