Global Practices
Big Data & Analytics Practice

A Lakehouse Implementation Using Delta Lake

Arun Viswanathan

GlobalLogic®
A Hitachi Group Company

Table of Contents

Abstract

A data lake is a centralized repository that enables a cost–effective storage of large volumes of data that provides a single source of truth (SOT). However, organizations face numerous challenges when using data lakes built on top of cloud-native storage solutions. These challenges include a lack of data consistency, unreliable data due to incomplete and corrupt files, performance issues, and the absence of schema enforcement and validation. One of the popular implementations of Lakehouse architecture is Databricks' Delta Lake which overcomes these challenges with an open-source storage layer built on top of existing data lake file storage formats such as Apache Parquet. We will first explore the differences between the architectures associated with data warehouses, data lakes, and lakehouses. Then, we take a glimpse under the hood to understand the inner workings of Delta Lake architecture. Finally, this white paper provides insights into how Delta Lake offers solutions to common problems encountered with data lakes such as ensuring data integrity with ACID transactions, providing scalable metadata management with distributed processing, data versioning with time travel, or preventing data corruption with schema enforcement.

Context and Problem

Data warehouses have existed for a long time to serve the needs of data analytics and business intelligence applications. They can however get very expensive as data volumes go up and are not optimally designed for handling unstructured data or semi-structured data. This led to the emergence of data lakes built on top of cloud object storage systems.

Data lakes provide a single source of truth for data and enable the cost-effective storage of large volumes of data. A data lake contains structured, semi-structured or unstructured data from many sources in both raw and processed formats. Data lakes, however, lack critical features provided by data warehouses such as support for transactions, enforcement of data quality, and the ability to mix appends and reads.

Data teams started combining data lakes and data warehouses into a **two-tier architecture** that is now dominant in the industry. In this data architecture, data is ingested into data lakes from different sources using ETL pipelines, and stored in low-cost object storage in formats compatible with common machine learning tools. A small part of the data is then ingested into a data warehouse using ETL pipelines for consumption by BI and reporting tools. In the white paper *"Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics"*[1], Ambrust, Ghodsi, Zin, and Zaharia explore the evolution of data platform architectures over a period of time. Figure 1 shows this evolution. This whitepaper also introduces the Lakehouse architecture from the perspective of the Databricks founders and provides a good background for the readers on the lakehouse concepts. A separate paper on Data Lakehouse will cover the core concepts of the data lakehouse architecture in more detail.

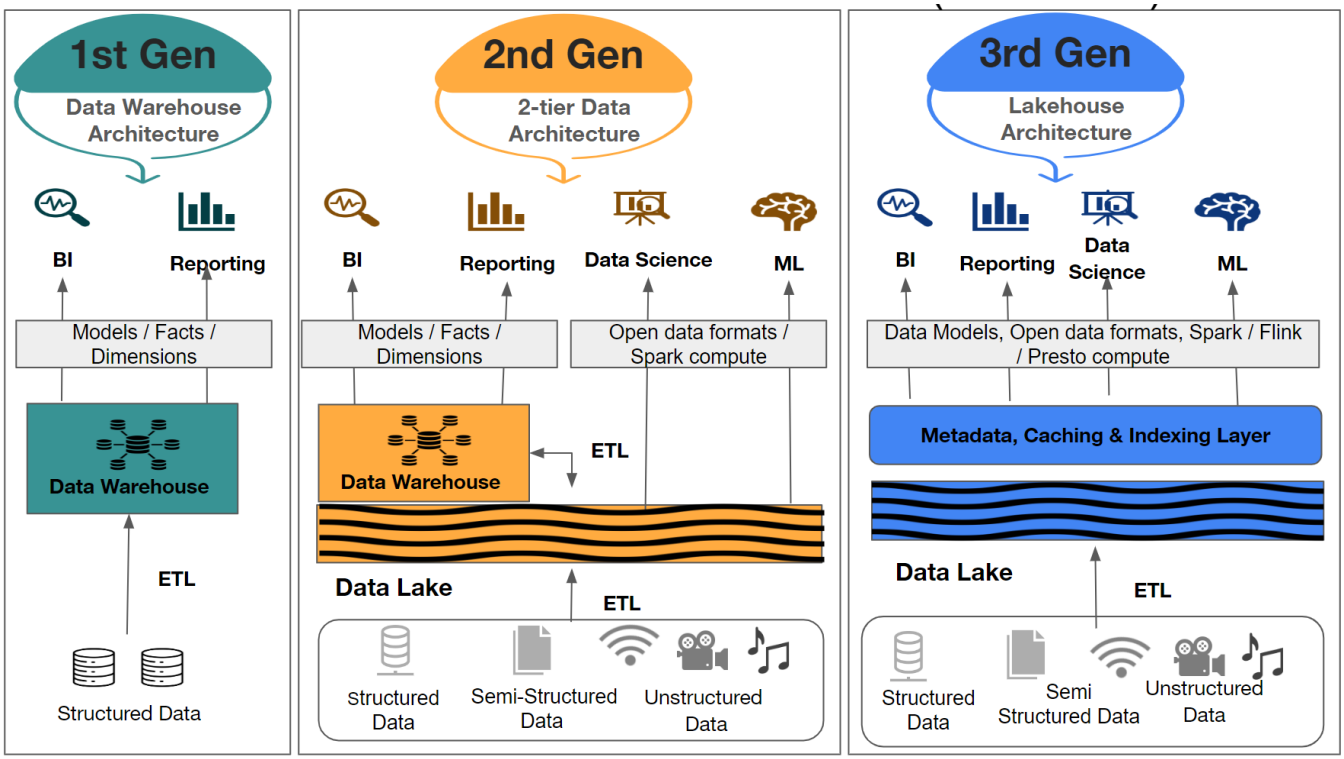

Figure 1 - Evolution of Data Platform Architectures (Source)

While the cloud-based data lake and data warehouse architecture is inexpensive due to separate storage (e.g., S3, ADLS) and compute (e.g., Redshift, Snowflake, Databricks), a two-tier architecture can be highly complex with the following challenges:

- **Reliability:** Maintaining consistency between the data lake and warehouse can be difficult and costly. Continuous engineering is required to perform ETL for data between the two systems and make it available to high-performance data analytics and BI
- **Limited support for advanced analytics:** The leading machine learning frameworks, such as PyTorch, XGBoost and TensorFlow, do not work well on top of warehouses. These systems need to process large datasets using complex non-SQL code and reading this data from the data warehouse via ODBC/JDBC is inefficient. Another related issue is the non-feasibility to directly access the internal warehouse proprietary formats.
- **Total cost of ownership:** Users also end up paying more for the continuous ETL jobs and additional storage cost for data copied to a warehouse. Additionally in commercial warehouses,

data is stored in proprietary formats that can increase the cost of data migration to other systems.

An alternative approach is to use the data lake to store data in open data formats directly for machine learning and analytics as well. When data lakes are build on top of cloud-native storage solutions, organizations face additional challenges that include data consistency, reliability, performance, and data quality:

- The data could be incomplete and contain corrupt files because the distributed file storage or cloud based storage solutions **don't support atomic transactions.** This can result in broken queries and failure of the related jobs.
- Cloud storage solutions are **not ACID compliant**. As a result, there is a lack of consistency when mixing appends and reads or when both batching and streaming data write to the same location.
- Another common issue with file storage is **file size inconsistency** i.e. files are either too small or too big. **Performance is adversely affected** with the existence of too many files and can require more time to access, open, and close files.
- **Lack of schema enforcement and validation** leads to data with inconsistent and low-quality structure. Mismatching data types between files or partitions can also cause transaction issues.
- Cloud storage solutions **do not support updates** to the data within the files. This requires the implementation of a custom strategy to handle updates.

Challenges with the existing data architecture leads us to consider and evaluate the **Lakehouse Architecture** pattern which combines the most advantageous features of data lakes and data warehouses into a single, integrated data management solution. It enables business intelligence and machine learning on all data by combining the flexibility, cost-efficiency and scalability of data lakes with the data management and ACID transactions of data warehouses. The Lakehouse architecture is based on open direct-access data formats, such as Apache Parquet and Apache ORC with a metadata layer on top to provide transactional views of the data lake and enable management features such as transactions, rollbacks to old table versions, and zero-copy cloning.

The Lakehouse is supported by a recent family of open source systems such as Delta Lake, Apache Iceberg, Apache Hudi and commercial products like Snowflake. The high level architecture for Lakehouse is similar across the tool stack. Delta Lake is an open source framework that is also available in the Databricks runtime with add-on proprietary features like Delta Engine, Photon, query optimizations, etc. In the figure 2 below we provide a representative comparison of the architectures across open source Lakehouse based on Delta Lake, Databricks Lakehouse based on Delta Lake and the Snowflake Data Platform.

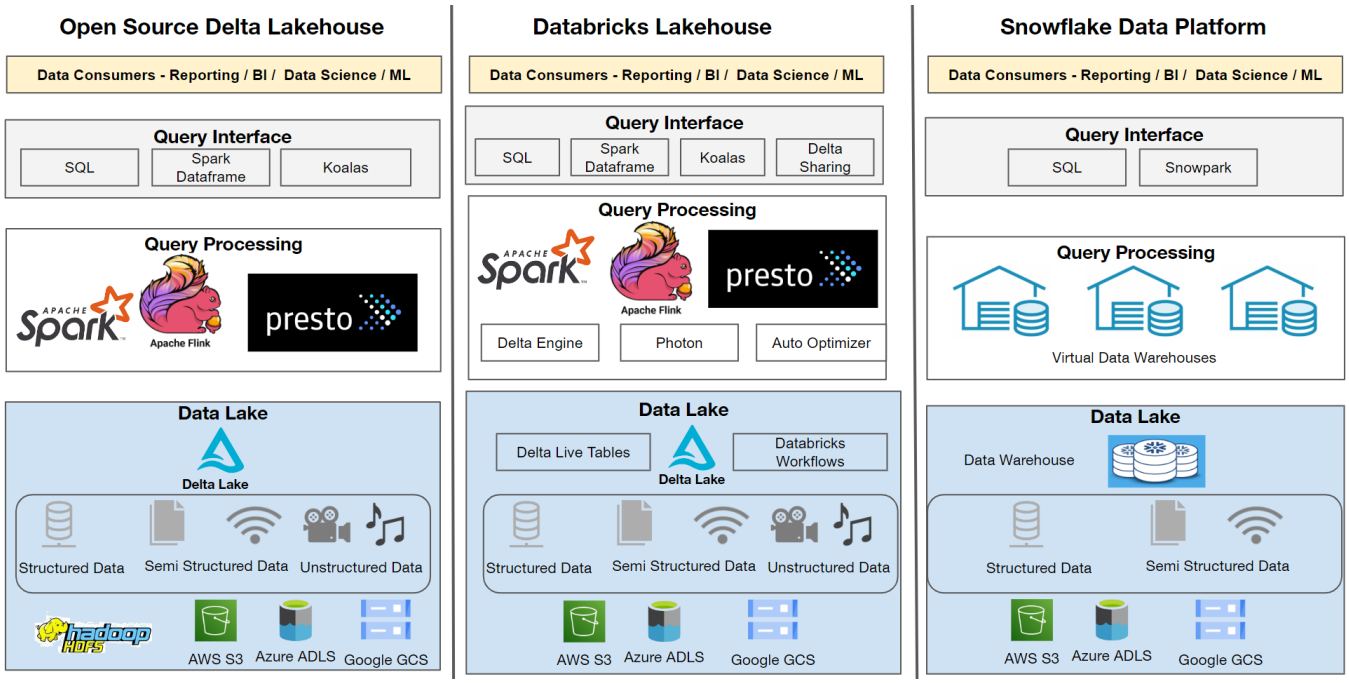

Figure 2 - Data Lakehouse Architecture Comparison (Source)

In this paper we explore **Delta Lake,** the open source storage layer that enables ACID transactions, petabyte scale metadata handling, and a unified streaming and batch data processing on top of existing data lakes, such as S3, ADLS, GCS, and HDFS.

Taxonomy

The taxonomies associated with Delta Lake require an understanding of some key terms. Data architects who collaborate with enterprise data stakeholders should be familiar with some key terms:

Terms	Description
Delta Lake	An open-source project originally developed by Databricks and now managed by the Linux Foundation.
ACID	ACID (Atomicity, Consistency, Isolation, Durability) refers to the set of four properties that define a transaction. Data storage systems that implement these operations are known as transactional systems. An ACID transaction is a database operation with these qualities. Each read, write, or modification of a table must adhere to the four fundamental properties, which ACID transactions ensure.
Data Lake	A central location for storing data of any scale, both structured and unstructured. It provides a single source of truth for the data and enables the cost-effective storage of large volumes of data.
Data Warehouses	Central data storage repositories that have combined data from different sources and are intended to facilitate BI and analytics.
Data	A data platform that merges the best aspects of data warehouses and data

Lakehouse	lakes into one data management solution. It combines the flexibility, cost-efficiency, and scale of data lakes with the data management and ACID transactions of data warehouses, enabling BI and ML on all data.
Transaction	Any operation that is handled as a single unit of work in databases and data storage systems that either completes completely or does not complete at all and leaves the storage system in a consistent state is referred to as a transaction.

Scoping Within the Enterprise City Map

Figure 3 illustrates an Enterprise City Map showing the high-level flow of information from different feature systems into the data platform.

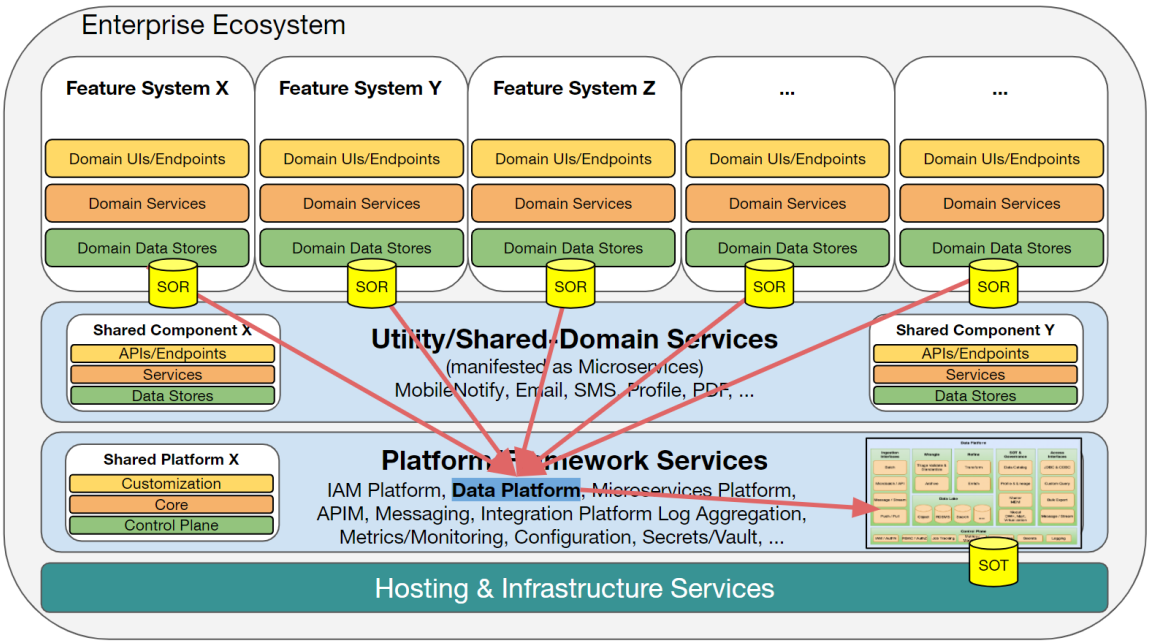

Figure 3 - Enterprise City Map (Source)

Scoping Within the Data Platform

WIthin the data platform, the Lakehouse capability is located in the Data Storage logical component which consists of transactional databases, object stores, search systems, etc. In Figure 4, the location of the delta lake lakehouse capability is highlighted in blue within the Enterprise Data Platform.

Multiple systems function as data producers. Data from all of these systems is moved into the delta lake by ingesting, wrangling, refining, and storing the data into the lakehouse. Data from the Delta Lake can be then leveraged and utilized by data consumers via access interfaces such as APIs and SQL endpoints.

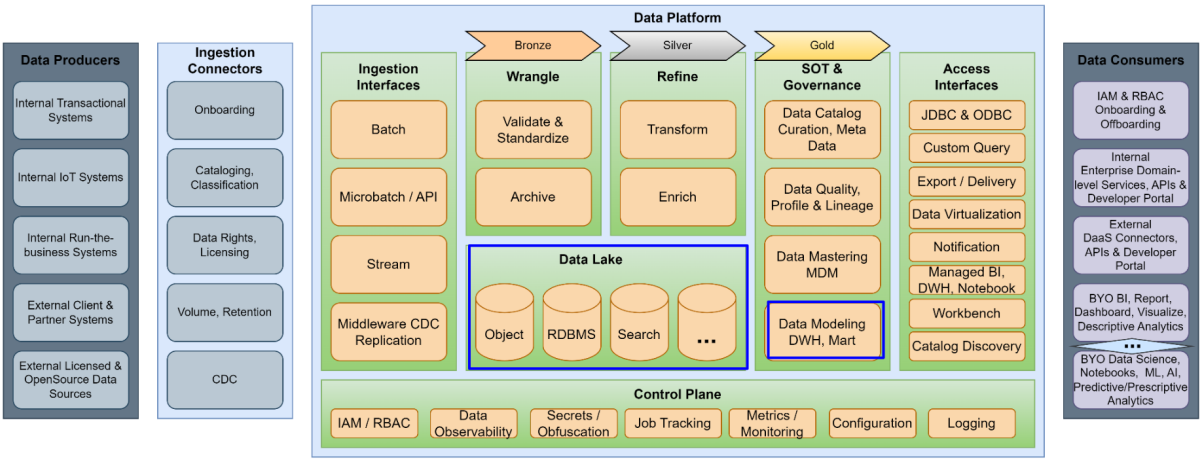

Figure 4 - Enterprise Data Platform (Source)

Implementing a Lakehouse Using Delta Lake

Delta Lake helps in building a Lakehouse architecture on top of existing data lakes as shown in Figure 5.

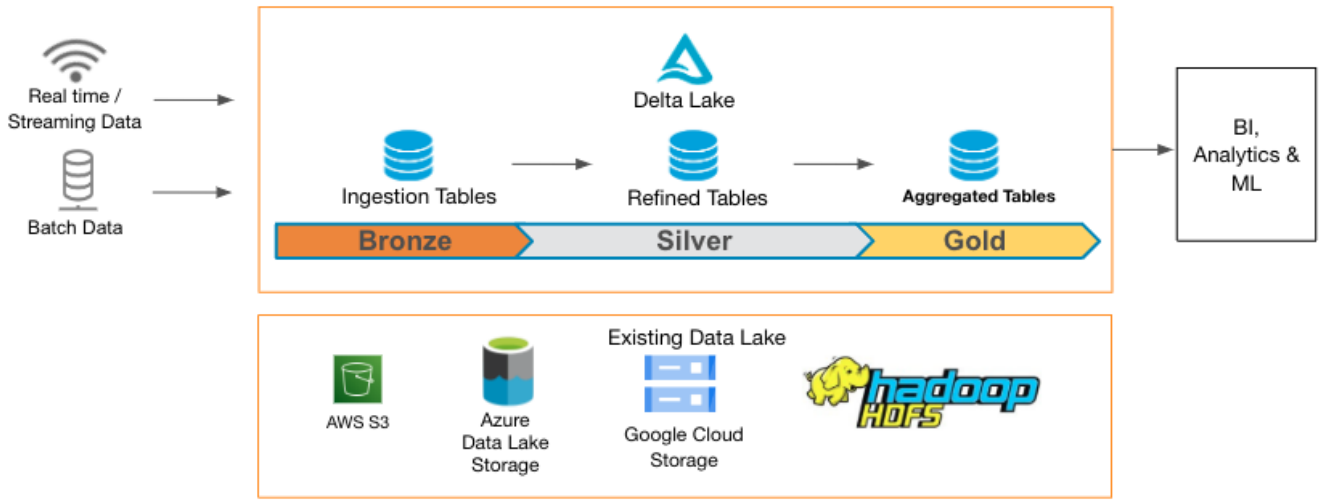

Figure 5 - Delta Lake Implementation (Source)

The Medallion architecture, which includes the Bronze, Silver, and Gold layers, is a common data integration approach used along with the Delta Lake.The ingestion layer, also known as the Bronze layer, is where data is first ingested and it helps create other downstream areas. The refined or Silver layer, which holds the cleaned, transformed, and curated data, forms the foundation for use in the final business layer. The final Gold layer stores the aggregated data that serves business needs such as BI reporting, data science, and ML. The Medallion architecture is addressed in detail in a separate paper here.

Delta Lake supports ACID transactions, scalable metadata handling, and a unified streaming and batch data processing. Next, let's take a look at these core functionalities in more detail.

Features of Delta Lake

Following are some key features offered by the Delta Lake framework:

Key Features	**Delta Lake's Support for Key Features**
ACID Transactions	Supports ACID transactions, that allow multiple queries and updates to be made to a table concurrently and consistently. Tracks commits made to the record directory and implements ACID transactions in a transaction log. Serializable isolation levels ensure data consistency across multiple users.
Time Travel	Tracks data version history, allowing users to access and revert (i.e. travel) to previous versions of the data as needed. Two options are available for accessing data versions: ● A timestamp or date string ● A version number
Scalable Metadata Management	Leveraging the distributed processing power of Spark, Delta Lake can handle the metadata for petabyte-scale tables consisting of billions of files.
Schema Management	Includes tools for data quality checks and data cleanup, helping users ensure that their data is accurate and consistent. Schema enforcement automatically validates and ensures that the data frame schema being written is compatible with the table's schema. A schema evolution tool enables the delta tables to automatically add new columns as soon as they are found.
Improved Performance	Includes features such as data skipping and predicate pushdown which can improve query performance.
Batch and Stream Processing	Integrates with both batch and stream processing systems (for example, Apache Spark Streaming), allowing users to build batch and real-time data pipelines.
Upserts and Deletes	Complex use cases such as change-data-capture, slowly-changing-dimension (SCD) operations, streaming upserts, etc. are supported through merge, update, and delete operations.
Delta Architecture	Allows the storage of Bronze (raw), Silver (refined), and Gold (aggregated) versions of data that can be used as the Single Source of Truth for multiple applications.
Integrations	Integrates with numerous compute engines including Apache Spark, Apache Flink, PrestoDB, Trino, and Hive. Also supports APIs for Java, Scala, Rust, Ruby, and Python programming languages.
Upcoming Features	In the latest release, Delta Lake offers advanced capabilities such as: ● Delta Universal Format to support Iceberg and Hudi table formats within Delta Lake ● Delta Kernel project (for building Delta connectors) ● Liquid Clustering-based partitioning (to address the shortcomings of Hive-style partitioning and current ZORDER clustering)

The core features of Delta lake are enabled by extending Parquet data files with a file-based transaction log for ACID transactions and scalable metadata handling. In the next section we will go under the hood to understand how Delta Lake achieves this.

Internal Architecture of Delta Lake

Delta Lake enables the Lakehouse Architecture by bringing a few key technology advancements that include:

- Adding Metadata layers on top of existing data lakes,
- Designing a new query engine that provide high-performance SQL execution, and
- Supporting an optimized and consolidated access for both BI and data science / machine learning tools

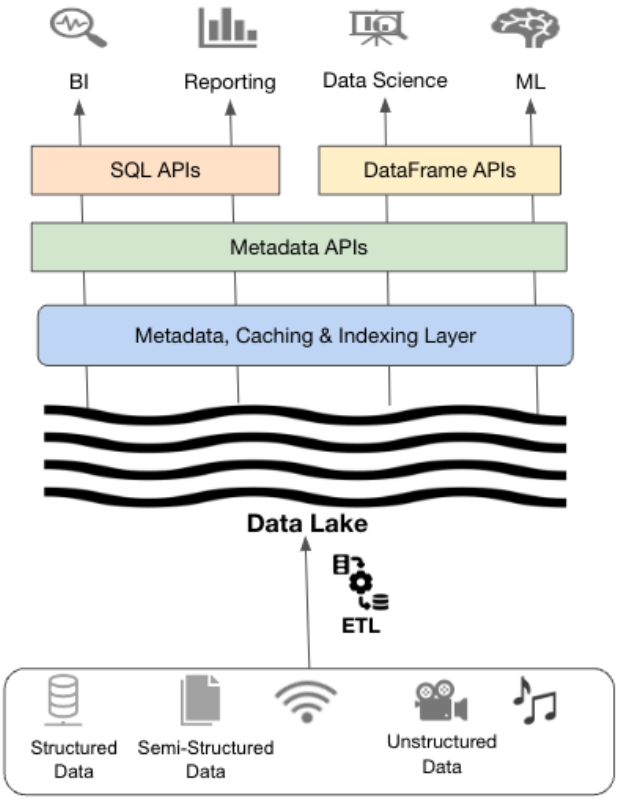

Figure 6 - Delta Lake Architecture Internals (Source)

Figure 6 shows the internals of the Delta Lake architecture. The **Metadata layers** sit on top of data files which use open file formats (for example Parquet) and keep track of the files. Support for streaming I/O (which does away with the requirement for message buses like Kafka), time travel to previous table versions, schema enforcement and evolution, as well as data validation, are all made possible by the metadata layers. On top of the Metadata layers are **metadata APIs** that provide a common interface for the **SQL API** and **dataframe APIs** to access the underlying data. The consuming applications such as BI and reporting tools or data science and ML programs consume the data through the SQL and Dataframe APIs.

Delta Lake uses a Transaction Log (also known as the **DeltaLog**) that stores all transactions that have been performed on a Delta Lake table from its creation.The DeltaLog provides a central repository that keeps track of all changes made to the tables by end users. In case of any updates made to the table, the transaction log of the table is updated with new changes. When a user runs a query on the delta table, this DeltaLog is read first by a processing engine like Spark to check the transactions posted to the table. This ensures that a user's version of a table is always synchronized with the master record as of the most recent query, and that users cannot make divergent, conflicting changes to a table.

The DeltaLog implementation also ensures support for ACID transactions within Delta Lake. As the transactions that execute fully and completely are recorded in the log, using that record as the single source of truth guarantees **atomicity** to Delta Lake. Whenever a user performs an operation (for e.g. INSERT, UPDATE or DELETE) to modify a table, the operation is broken down into a series of steps based on one or more of the actions like add/remove file, metadata update, set transaction, change protocol or commit info. These actions are then recorded as ordered, atomic units known as commits. So whenever a Delta Lake table is created, a *_delta_log* subdirectory is created under that table directory to store the transaction logs. Any changes to the table are recorded as ordered, atomic commits in the transaction log. Delta Lake periodically also generates checkpoint files which help with enhanced read performance. The checkpoint files include the entire state of the table at a point in time saved in native Parquet format. Figure 7 shows the sample contents of the delta lake table.

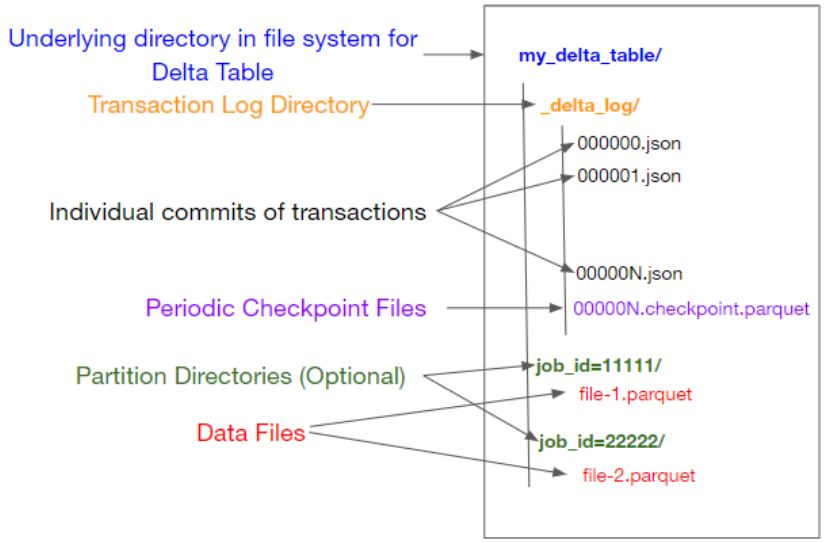

Figure 7- DeltaLog Transaction Logs (Source)

For **concurrency**, Delta Lake employs an approach known as optimistic concurrency control. For concurrent transactions, Delta Lake goes with the assumption that changes made to a table by different users can be combined without any conflicts. This approach ensures very high performance when dealing with petabytes of data. Delta Lake handles this by implementing the rule of mutual exclusion for figuring out how commits should be ordered (also known as serializability in databases), and determining a path of action if two or more commits are made concurrently. This rule ensures that the final state of the table whether after multiple concurrent writes or serial writes remains consistent

thus delivering on the ACID principle of **isolation.** Finally, since all of the transactions made on Delta Lake tables are stored directly to underlying cloud storage which offer the highest **durability**.

Since the transaction log provides a detailed set of instructions on what changes were done to the table since it was created, we can easily recreate the state of a table at any point in time by starting with an original table, and processing only commits made prior to that point. This ability enables the versioning of data or what is known as "**time travel**".

Schema enforcement or schema validation feature rejects data being written to a table that does not match the table's original schema thus ensuring data quality. Delta Lake uses an approach called schema validation on write where all new writes to a table are checked for compatibility while writing into the table. In case of incompatibility in the schema the entire transaction is canceled, and an exception is raised to the user regarding the mismatch.

For tables which expect schema to change over a period of time, the **Schema evolution** feature provides that functionality. By enabling the mergeSchema option, the schema of the table can automatically adapt to include one or more new columns while performing an append or overwrite operation. Delta Lake supports added nested fields as well to the schema by adding these fields to the end of their respective struct columns.

Benefits of Delta Lake

In this white paper, we explored the core functionalities provided by Delta Lake and how they are implemented. These core functionalities of Delta Lake provide the following benefits for the lakehouse architecture:

1. **Increasing data freshness for data analytics –** The lakehouses strategy enables enterprises to access the most recent and freshest data for use in BI, reporting, and insights, in contrast to the 2-tier data architecture approach which may contain outdated data. As a result, organizations can make decisions based on the most recent data.
2. **Self-service data experience** – Organizations can add more value by giving users faster access to data by implementing data lakehouses in their operations. Organizations can empower users to do tasks without a high level of technical expertise by leveraging the power of high-performance queries, more accessible data access, flexibility, and the use of data management principles. Additionally, end users have a better user experience with performance optimization capabilities such as caching hot data in RAM and SSDs, data layout optimizations to cluster co-accessed data, auxiliary data structures such as statistics and indexes, and vectorized execution on modern CPUs.
3. **Unified cloud data lakehouse management** – The data landscape is made simpler by combining the capabilities of the data warehouse and data lake, and data professionals no longer need to move data often between various platforms. Instead, all data management tasks are carried out within a single system, which is simpler to maintain and requires less time and effort.
4. **Layered data architecture** – The data lakehouse solution recommends a three-layered approach to handling data coming from producers to consumers. These layers are raw, refined, and aggregated. Because different ETL procedures take place before the data enters the curated layer, this layered data architecture helps to increase data quality.

5. **No data lock-ins** – Data lakehouses utilize open storage formats like Parquet file format for easy storage and retrieval, which allows easy interoperability among different systems. Hence, organizations can quickly build their data lakehouse by leveraging these tools. Utilizing open data formats means no data lock-ins, as there is a free exchange of data where rules occur by using an open decision-making process.
6. **Achieve compliance -** Companies are required by laws like the General Data Protection Regulation (GDPR) and the California Consumer Protection Act (CCPA) to delete consumer data upon request from the person whose information it is. In a typical Parquet data lake, updating or deleting data requires a lot of computing power. Contrarily, Delta Lake offers DELETE and UPDATE operations for simple table data editing.

Conclusion

This white paper first explored the challenges with existing data warehouses and data lakes based architectures and how lakehouse architecture provides an alternative solution. Delta Lake is a popular framework that supports the lakehouse architecture, and brings additional reliability and performance features to the data lake. It provides a number of features that ensure accurate and consistent data in the data lake. Delta Lake also supports different compute engines including Spark, PrestoDB, Flink, Trino, Hive and provides APIs for integration through Scala, Java, Rust, Ruby, and Python. Overall, Delta Lake can be a useful tool for building a Lakehouse Architecture by adding transactional capabilities to data stored in Apache Parquet files. But it is not a one-size-fits-all solution and may not be the best choice for every situation. The Data Architect should carefully weigh in all the strengths and weaknesses of the platform within the context of an organization's requirements and make an informed decision.

References

1. Armbrust et al. Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics
2. Armbrust et al. Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores. PVLDB, 13(12): 3411-3424, 2020
3. What is a Data Lakehouse?
4. Delta Lake
5. Delta Lakes: A Comprehensive Guide 101
6. Demystifying Delta Lake. In the real data world, the majority of… | by Saurabh Mishra | Analytics Vidhya | Medium
7. Diving Into Delta Lake: Unpacking The Transaction Log
8. Top 5 Reasons to Convert Your Cloud Data Lake to a Delta Lake - The Databricks Blog