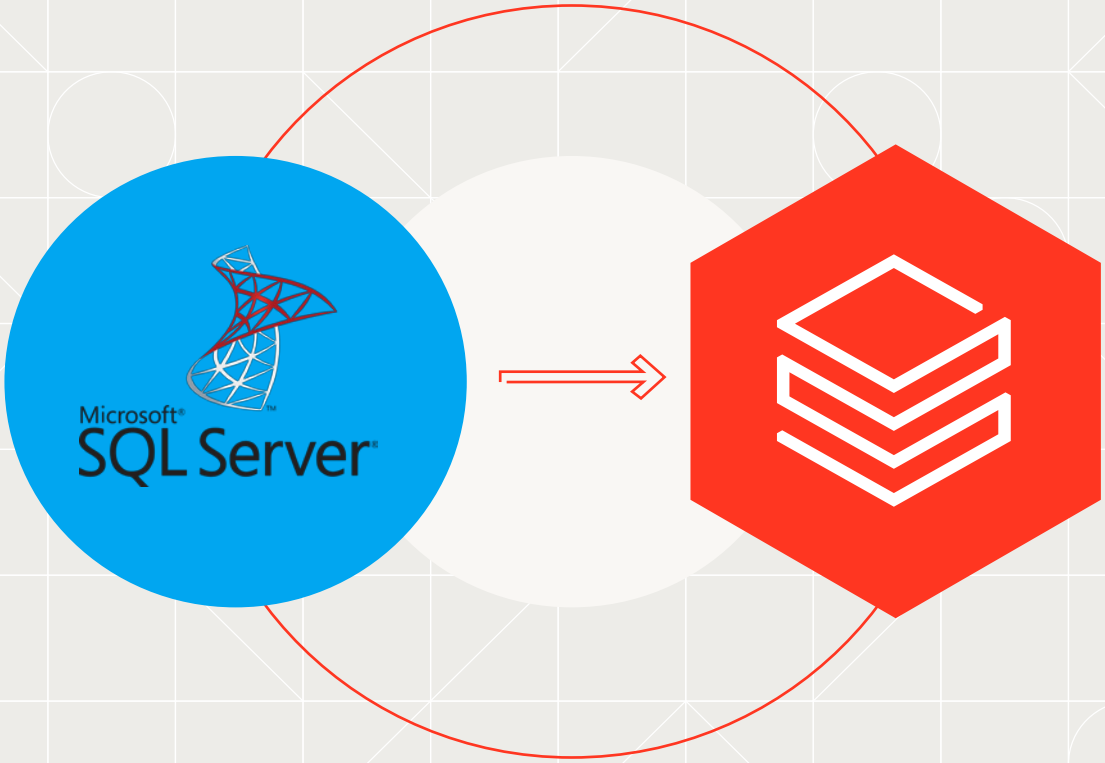
databricks

Guide

Microsoft SQL Server to Databricks Migration Guide



databricks

Contents

Introduction

Traditional enterprise database warehouse (EDW) appliances like Microsoft SQL Server come with significant limitations — high costs, lack of support for unstructured data, built-in AI, ML or real-time streaming capabilities and scaling storage or computing that is challenging and expensive. To solve this, enterprises have different data marts, data warehouses, data lakes, ML platforms and streaming platforms that create silos as it requires constant ETL processes to move data between platforms for different workloads, increasing complexity and slowing down insights.

Databricks' Data Intelligence Platform introduces a paradigm shift by eliminating the need for separate data processing systems and constant data movement. Instead of copying data between warehouses, marts, lakes and ML platforms, Databricks brings different processing engines to a single copy of data in the cloud, enabling seamless data warehousing, AI, ML and GenAI use cases on a single platform and data asset. With its lakehouse architecture, Databricks provides governance, scalability and advanced analytics while decreasing costs and operational complexity. Migrating to Databricks ensures your organization is ready for the future with a unified, efficient and intelligent data platform.

ABOUT THIS GUIDE

This guide provides a detailed roadmap for migrating data warehouse workloads from SQL Server to the Databricks Data Intelligence Platform. It outlines key differences between the two systems, standard data and code migration patterns, and best practices to streamline the transition.

Additionally, it compiles proven methodologies, tool options and insights gained from successful migrations. This migration guide covers theoretical concepts and practical applications and is a comprehensive resource for organizations looking to leverage Databricks for enhanced performance, scalability and advanced analytics.

MIGRATION STRATEGY

Successfully migrating from SQL Server to Databricks requires meticulous planning, strategic alignment and precise target architecture to secure a favorable outcome. Following a proven structured migration methodology is critical to achieving a seamless, effective migration to Databricks, enabling your organization to realize value and position itself for rapid future innovation.

OVERVIEW OF THE MIGRATION PROCESS

Proper planning is required to migrate data and ETL processes from legacy on-premises systems to cloud technologies. This migration involves transferring data and business logic from on-premises infrastructure to the cloud.

Despite the substantial differences between SQL Server and Databricks, there are surprising similarities that can facilitate the migration process:

- Despite its proprietary SQL dialect, Microsoft SQL Server's T-SQL adheres mostly to ANSI SQL standards, providing compatibility with Databricks SQL syntax.
- Code migration can be accomplished through code refactoring, leveraging the shared ANSI SQL compliance between the two systems.
- Fundamental Data Warehouse concepts exhibit similarities across SQL Server and Databricks.

The migration process typically consists of the following technical implementation phases:

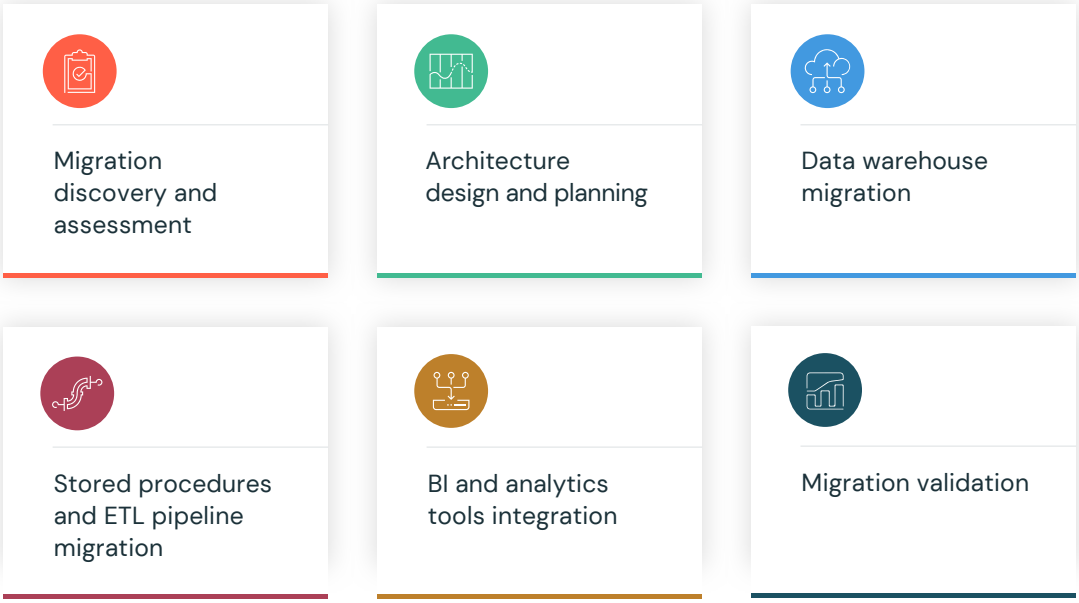


Figure 1: Technical migration approach

Phase 1: Migration Discovery and Assessment

Conducting a migration assessment is crucial before migrating any data or workloads. This enables Databricks to:

- Gain insight into data ingress and egress, ETL patterns, data volume, orchestration tools and execution frequency.
- Understand the technologies involved in upstream and downstream integrations.
- Assess the business criticality and value of the existing systems.
- Evaluate the existing security framework and access control mechanisms.
- Gather pertinent information to provide a realistic estimation of the effort required for migration.
- Compare and calculate infrastructure costs.
- Identify any imminent deadlines, particularly regarding license renewal fees for the existing SQL Server setup.
- Document any functional or cross-functional dependencies in the migration plan.

Databricks recommends automation tools, such as the recently acquired BladeBridge Code Analyzer, to expedite the gathering of migration-related information during this phase.

Typically, these profiling tools examine SQL Server system usage via its system views and catalog tables, providing consumption and complexity insights and an inventory of objects and code migration complexity. They capture the types of workloads, long-running ETL queries and user access patterns. This level of analysis aids in pinpointing databases and pipelines that contribute to high operational costs and complexity, thereby supporting the prioritization process.

Our BladeBridge Code Analyzer not only classifies queries based on their complexity in "T-shirt sizes" (small, medium, large, extra-large, etc.) — but also assesses function compatibility of Bteq scripts and stored procedures, which is vital in ensuring seamless migration.

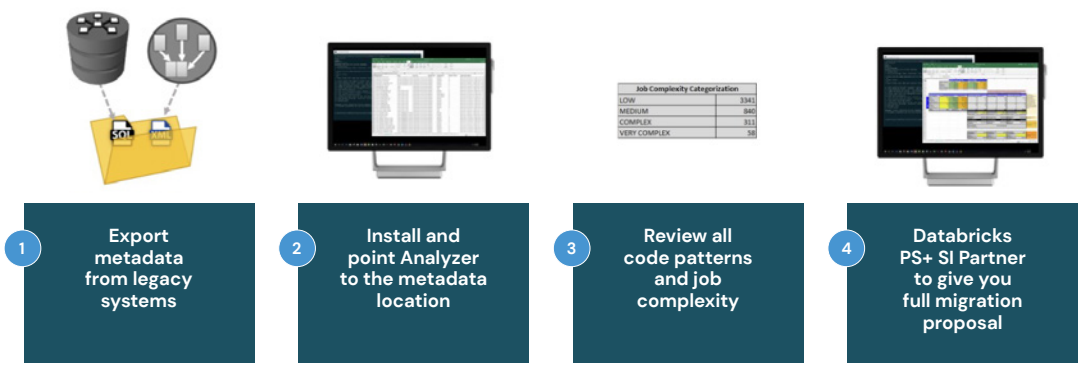


Figure 2: Running Databricks migration analyzer

Phase 2: Architecture Design and Planning

SQL Server and Databricks operate in markedly different ways. SQL Server requires careful selection of a suitable Primary Index with high cardinality to ensure proper data distribution across all participating.

By contrast, the Databricks Intelligence Platform is a distributed system by design; the data distribution depends on the configuration of a cluster and the nature of the data. For example, if data is loaded from a sample CSV file into a Databricks Notebook using the spark.read.csv() function, the data will be automatically distributed across the nodes in a cluster. By default, Spark will split the data into partitions, processing each partition by a separate task on an individual node. This allows for efficient parallel processing of the data.

The Databricks distributed design facilitates horizontal scaling, enabling data distribution and computations across multiple nodes in a cluster. This capability allows Databricks to process large datasets and handle high query volumes efficiently, surpassing the capabilities of a traditional database system like on-premises SQL Server.

It is essential to consider these distinctions when migrating from SQL Server to Databricks. By consciously mapping the similarities and differences between the two platforms, organizations can better understand Databricks' capabilities with SQL Server.

MONITORING AND OBSERVABILITY FRAMEWORK

Plan for a comprehensive monitoring and observability framework that provides real-time insights into the cloud infrastructure and applications' performance, health and security.

EDW ARCHITECTURE

In legacy EDW architectures, data from various systems is typically ingested via ETL tools or ingestion frameworks. After landing in the raw layer, the data progresses to the stage or central layer, where further cleansing and processing occur. Finally, it moves to the last layer, containing the most complex business logic.

After the final layer is prepared, use it for reporting purposes through third-party tools such as Microstrategy or BusinessObjects.

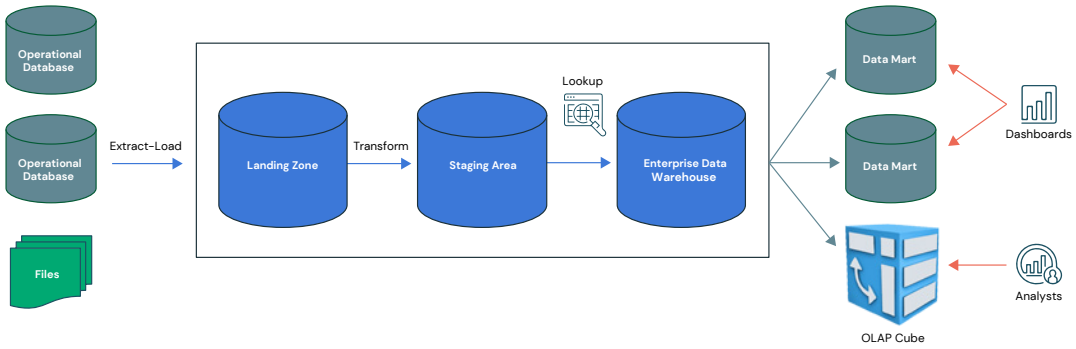


Figure 3: SQL Server reference architecture of an enterprise data warehouse

It is imperative to analyze the current architecture, the as-is architecture comprehensively. This involves understanding upstream and downstream integrations and the respective tools and technologies.

Following this analysis, assess the potential for modernizing each stage of the target architecture. This entails evaluating how well an organization can transition from legacy systems to modern alternatives at each stage. Key decisions on data ingest into cloud storage include evaluating where features like Databricks Autoloader, Lakeflow Connect or Lakehouse Federation can be implemented. ETL modernization and partners are considered, and then BI tool compatibilities are also done. A target architecture and tooling roadmap is created that guides the migration process.

Below is an example of a data warehousing architecture on Databricks with various ISV partner integration options.

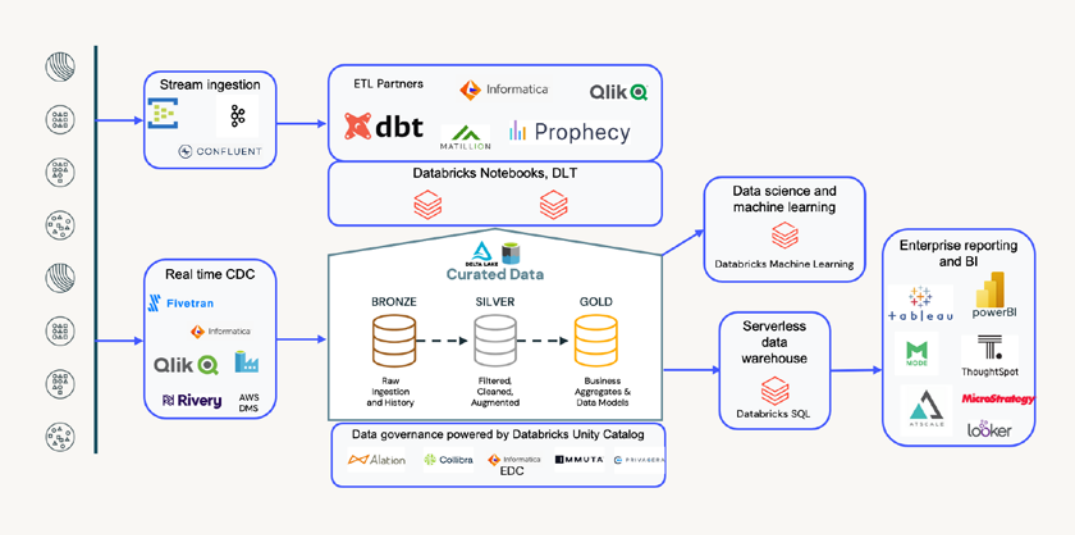


Figure 4: Modern data warehousing on Databricks

Following the architectural alignment, we will dive deeply into the SQL Server's current features.

SQL SERVER VS. DATABRICKS FEATURE MAPPING EXAMPLE

OBJECTS/ WORKLOAD	SQL SERVER	DATABRICKS
Compute	SQL Server on-premises compute	Databricks Managed Clusters optimized for workload types with a runtime: • All-purpose for interactive/developer use • Job clusters for scheduled pipelines • SQL warehouse for BI workloads
Storage	Physical HDD or SSD for on-premises deployment.	Cloud storage (Amazon S3, Azure Blob Storage, Azure Data Lake Storage Gen2, Google Cloud Storage)
Tables	SQL Server Tables	Delta tables in Unity Catalog
Format	SQL Server proprietary	Delta and Iceberg Format (open source)

OBJECTS/ WORKLOAD	SQL SERVER	DATABRICKS
User Interface	SQL Server Management Studio (SSMS)	Databricks workspace Databricks collaborative notebooks Databricks SQL Query Editor Databricks CLI Visual Studio Code Spark Connect Databricks API Terraform
Database Objects	Tables, Views, Materialized Views (Join Index), Stored Procedures, UDFs	Tables, Views, Materialized Views, DLT, UDFs
Metadata Catalog	Built-in system tables under the DBC schema	Unity Catalog
Data Sharing	No native support for on-premises mode	Delta Sharing Delta Sharing Marketplace
Data Ingestion	Bulk Insert/Bulk Load SQL Server Integration Services (SSIS) OPENROWSET Linked Servers	COPY INTO CONVERT TO DELTA Auto Loader DataFrameReader Integrations via Partner Connect Add data UI
Data Types	T-SQL Data types	Data Types in Databricks
Workload Management	SSIS - SQL Server Integration Services Resource Governor	Cluster configuration (policies) Multi-clustering Intelligent Workload Management Intelligent Autoscaling Adaptive Routing
Security	RBAC Database roles Database object permissions Dynamic Data masking (2016 and above) Row-level security Column level security	IAM, RBAC Database object permissions (UC) Dynamic data masking Row-level security Column level security
Storage Format	SQL Server Proprietary	Delta (Parquet files with metadata) and Iceberg format

OBJECTS/ WORKLOAD	SQL SERVER	DATABRICKS
Sorting	Unsupported	Z-Ordering Liquid Clustering
Programming Language	SQL only	SQL, Python, R, Scala, Java
Data Integration	External ETL tools like SSIS, PowerCenter	DLT Databricks workflows External tools (dbt, Matillion, Prophecy, Informatica, Talend, etc.)
Orchestration	SSIS - SQL Server Integration Services	Databricks Workflows
Machine Learning	Unsupported	Databricks ML (Runtime with OSS ML packages, MLflow, feature store, AutoML)
Pricing Unit	PerCore Licensing	Databricks units (DBUs)

The table above compares key features between SQL Server and Databricks. Undertaking a thorough comparison is essential during this stage of the migration process. This systematic process ensures a comprehensive understanding of the required transformation, facilitating a smoother transition by identifying equivalent services, functionalities and potential gaps or challenges.

Typically, by the end of this phase, we have a good handle on the scope and complexity of the migration and can come up with a more accurate migration plan and cost estimate.

Phase 3: Data Warehouse Migration

CONSIDERATIONS FOR WORKSPACE CREATION

The entire planning of a new Databricks environment for a customer is out of scope for this document (please see Azure Databricks Administration Guide | Databricks on AWS administration introduction | Databricks administration introduction on GCP and Azure Databricks well-architected framework | Databricks on AWS well-architected data lakehouse | Introduction to the well-architected data lakehouse | Databricks on Google Cloud) for a much more in-depth guide for planning, deploying and configuring a customer Databricks environment), but key considerations for the target workspaces of a data warehouse migration include:

- **Separation of environments:** Requiring different workspaces for development, staging, production and other environments.
- **Separation of business units:** Different workspaces can be designed for different departments, such as marketing, finance, risk management, etc. BU-separated workspaces can make chargeback billing models easier instead of relying on tags.

Note: If these separate departments must share data and collaborate, features like delta sharing and Unity Catalog will help.

- **Implementing modern data architectures:** Different workspaces are required to support modern data architectures, such as Data Mesh architecture, to decentralize data ownership for different domains.

CONSIDERATIONS FOR SCHEMA AND DATA MIGRATION

Once the Databricks workspaces have been established, the initial migration phase involves migrating schema and data. This includes metadata, such as table data definition language (DDL) scripts, views and table data.

As organizations navigate the process of migrating data out of SQL Server, it's crucial to consider several key decisions. These include:

- What is the target design for the tables being migrated?
- Should the destination retain the same hierarchy of catalogs, databases, schemas and tables?
- How can the separation into hot (recent) and cold (historical) datasets be considered to optimize data migration?
- Consider a potential cleanup or reorganization of the existing data footprint in SQL Server. This step could significantly enhance the efficiency and effectiveness of the migration process, reducing potential issues, simplifying data management and optimizing resource use in the new environment.

There are also a few recommendations that can help to enable a smoother and less risky migration:

- Data modeling: As part of the migration, there might be a need to refactor or reproduce a similar data model in an automated and scalable fashion. Visual data modeling tools like Quest ERWIN or SQLDBM can be found in Databricks Partner Connect. These tools can help accelerate the development and deployment of the refactored data model in just a few clicks. Such a tool can reverse engineer a SQL Server data model (table structures and views) in a way that can be implemented in Databricks easily.
- When migrating DDLs, verifying the provenance of the data schema (e.g., source data) is essential. Consider an instance where one of the SQL Server tables is presented with a data type that is proprietary to SQL Server (Ex. cast, convert, etc.). That exact data type might not find a precise equivalent for such data types in Databricks, so replacement data types are needed.
- Schematic and data migration can proceed after carefully deliberating and finalizing key decisions. It's generally advisable to avoid introducing significant changes to the schema structure during migration.

Delta Lake's Schema Evolution capability allows for flexible modification and schema evolution after data is placed in Delta, simplifying the initial migration process. This approach facilitates easier data comparison in SQL Server and Databricks during parallel runs.

Maintaining the existing schema during migration ensures consistency, easing the data verification process and fostering a more reliable and efficient migration.

RECOMMENDED APPROACH

It is important to note that not every data migration will follow the same pattern, as each migration is influenced by unique factors such as data volume, system complexity and organizational requirements. However, Databricks recommends adhering to the following general flow for an effective and efficient data migration process:

1. Migrate enterprise data warehouse (EDW) tables into Delta Lake medallion data architecture:
   - The raw layer into Bronze
   - The stage/central or historical layer to Silver
   - The final layer or semantic layer to Gold
2. Migrate or build data pipelines that populate the Bronze, Silver and Gold layers within the Delta Lake incrementally.
3. Backfill of Bronze, Silver, and Gold tables as needed.

For more details on modeling approaches and design patterns, refer to Data Warehouse Modeling Techniques.

PHASE 3.1: SCHEMA MIGRATION

Before offloading tables to Databricks, it's essential to establish the schema of the tables within the Databricks environment. These can be beneficial if an organization possesses Data Definition Language (DDL) scripts from its existing system. With minor adjustments, mainly to accommodate the data types used in Databricks, these scripts can be utilized to create corresponding table schemas in Databricks. Reusing existing scripts can streamline the preparation process, thus fostering efficiency and maintaining schema consistency during the migration.

DDL scripts can also be extracted from SQL Server using the SQL Server Management Studio tool or SQL Server Management Objects (SMO) API or constructed dynamically using metadata available in system tables.

Once the DDLs are extracted, converting them to comply with Databricks is essential. Below are some of the example scenarios to be considered:

1. Namespace mapping, i.e., schema/object in SQL Server to catalog/schema/object in Databricks Unity Catalog.
2. Databricks supports IDENTITY columns only on bigint columns.
3. Table options such as distributions and indexes are not applicable in Delta tables.

4 | Caution must be exercised when converting PRIMARY and SECONDARY indices to partitions in Delta tables. Over partitioning can lead to unnecessary overhead and minor file problems, ultimately compromising performance in the Lakehouse architecture. Delta's default partition size is 1TB, and Z-order indexes and predictive optimizations simplify the design in Databricks.

5 | Additional Delta table properties can be specified via the TBLPROPERTIES clause, e.g., delta.targetFileSize, delta.tuneFileSizesForRewrites, delta.columnMapping.mode, and others.

PHASE 3.2: DATA MIGRATION

Transferring legacy on-premises data to a cloud storage location for seamless consumption in Databricks can be a demanding task, but there a few viable options:

1 | **Databricks Lakeflow Connect** offers fully managed connectors for ingesting data from SaaS applications and databases into a Databricks lakehouse. For more information, refer to Ingest data from SQL Server.

2 | **Leveraging Databricks Lakehouse Federation:** Databricks Lakehouse Federation allows for federated queries across different data sources, including SQL Server.

3 | **Microsoft Azure Data Factory (ADF):** Another approach involves using Azure ADF to extract data directly from SQL Server using a JDBC/ODBC connector and then store it in a Blob.

4 | **ISV Partners such as Qlik Replicate:** Qlik can replicate data from SQL Server to the Databricks Delta table for historical and CDC data.

5 | **Using Databricks' JDBC Connector:** Databricks provides a JDBC (Java Database Connectivity) connector that facilitates direct reading from SQL Server databases.

PHASE 3.3: OTHER DATABASE OBJECTS MIGRATION

Other Database Objects, such as Views, Stored procedures, and Macros and Functions can also be easily migrated to Databricks via our automated code conversion processes. Please review this helpful cheat sheet packed with essential tips and tricks to help users start on Databricks using SQL programming in no time! Some key pointers while converting T-specific SQL objects:

- **Views** is typically used for data access control. In the context of the medallion architecture, they can be considered part of the Gold layer. Views would also be used as an intermediate data structure while transforming data and publishing the business KPIs to final users.

- **Stored procedures** are typically used in data warehouse environments to leverage the ELT pattern. This methodology signifies that most data processing transactions are performed in the warehouse.
- **Functions** are generally utilized to execute additional transformations over scalar values. Inline table-valued functions, while not extensively used, hold their significance. When called upon, they can be conceptualized as parameterized views, offering a more dynamic approach to data processing. They can adapt to varying input parameters, providing greater flexibility in handling and transforming data.

Stored procedures implementation in Databricks

Note that with the newly released Databricks SQL Scripting support, you can now easily deploy or convert powerful procedural logic within Databricks. Databricks SQL scripting supports compound statement blocks (with BEGIN....END). Within the Databricks SQL scripting procedures, we can declare local variables, user-defined functions, use condition handlers for catching exceptions and use flow control statements such as FOR loops over query results, conditional logic such as IF and CASE and means to break out loops such as LEAVE and ITERATE. These features make stored procedures (T-SQL) migration to Databricks even easier.

Implement Slowly Changing Dimensions

Slowly Changing Dimensions (SCDs) are crucial in data warehousing and managing historical changes to dimensional data over time, including updates, inserts or deletions to maintain accurate historical data. However, migrating SCDs from SQL Server to Databricks poses challenges, such as ensuring data consistency and accuracy, especially with large data volumes. Additionally, mapping SCD logic from T-SQL to Databricks SQL requires careful consideration and testing to maintain functionality and performance. Overall, effectively managing and migrating SCDs from SQL Server to Databricks demands thorough planning, testing and optimization for a successful transition.

Here are a few resources that can assist in achieving a successful transition:

- How to implement SCDs when you have duplicates - Part 1
- How to implement SCDs when you have duplicates - Part 2: DLT
- APPLY CHANGES API: Simplify change data capture in DLT

PHASE 3.4: DATA SECURITY MIGRATION

When discussing security migration, we need to consider both authentication and authorization.

When planning SQL Server security objects migration, it is essential to understand the differences between the two platforms and correctly map SQL Server security capabilities to Databricks Data Intelligence Platform security capabilities.

Authentication

MS SQL Server supports Active Directory (AD) and SQL Server-specific login and password authentication.

Azure AD can connect with on-premises AD (users, groups, service principals) registered in the Azure Databricks account and used across Databricks workspaces, Unity Catalog and within Databricks SQL.

SQL Server SQL logins are used in SQL basic authentication, defined using CREATE LOGIN statements. On the other hand, Azure Databricks SQL supports only Azure AD authentication. This means that SQL logins in SQL Server must be converted to Azure AD principals for migration purposes or wholly excluded from the migration scope.

Note that when discussing MS SQL Server, it is natural to migrate to Azure Databricks. Therefore, authentication in Databricks on AWS is not within the scope of this guide.

Authorization

Though Databricks supports Hive metastore, this document will focus on Unity Catalog only, as it is a future-proof approach for data governance in Databricks Data Intelligence Platform.

From a security perspective, Unity Catalog shares many similarities with SQL Server. Both offer authorization models where permissions are assigned to objects and use ANSI-compliant SQL statements such as GRANT and REVOKE. However, it is crucial to understand the difference between the two to execute a successful migration.

First, SQL Server offers a concept of database roles — another type of database principals that may have members apart from assigned permissions. Thus, all members get all permissions assigned to their respective roles. At this time, Databricks doesn't have a concept of database roles. The most suitable migration strategy is leveraging Unity Catalog Account groups.

Similarly, Databricks doesn't have fixed database roles, which are widely used in SQL Server (*e.g., db_owner or db_securityadmin*). We recommend revising the usage of such roles and moving to a more granular permissions model in the Unity Catalog.

Both SQL Server and Databricks offer similar GRANT and REVOKE statement syntax. However, Databricks doesn't possess DENY statements. Therefore, there is no explicit support for the denial of permission. If you use this security technique in your design, we suggest revising and redesigning to an approach based on explicit grants and, optionally, permission inheritance.

It is also essential to understand the difference between permissions and privileges that can be assigned to different database objects. One of the most important differences is that SQL Server offers INSERT, UPDATE and DELETE permissions for tables, while Databricks offers only MODIFY permission, which essentially includes all three permissions.

A more detailed description of the permissions applicable to specific securable objects techniques is available in the documentation on the hyperlinks that follow:

- Databricks
- SQL Server

We recommend following these standard practices in Databricks Unity Catalog to enable better permissions manageability and operational excellence.

- Assign permissions at the highest possible level (e.g., schema, catalog) to leverage permissions inheritance.
- Assign permissions to groups rather than individual principals.
- Using AAD groups to manage permissions rather than giving permissions to individuals is recommended.

Column-level security, **Row-Level Security**, and **Dynamic Data Masking** are implemented via Databricks dynamic views, row-level filters, and column masking.

Phase 4: Code and ETL Pipelines Migration

A comprehensive understanding of the pipelines, extending from data sources to the consumption layer and including governance aspects, is a crucial prerequisite for executing an effective workload migration. It's not just about moving data; it's about ensuring a smooth, efficient transition and maintaining data integrity. Data pipeline migrations are multifaceted operations, especially those transitioning from SQL Server to Databricks. They encompass several key components: orchestration, source/sink migration, query migration and refactoring. These elements contribute to a successful and seamless data migration process.

Depending on the adopted data ingestion and transformation pattern (ETL vs. ELT), more migration efforts may be needed for either SQL query migration or ETL pipeline redesign.

ORCHESTRATION MIGRATION

An ETL orchestration can refer to orchestrating and scheduling end-to-end pipelines covering data ingestion, data integration, result generation or orchestrating DAGs of a specific workload type like data integration. In SQL Server, the orchestration is typically done using SQL Server Agent or SSIS, and there are generally these options when migrating these workflows:

1. Use Databricks Workflows to orchestrate the migrated pipelines. Databricks Workflows support various tasks like Python scripts, Notebooks, dbt transformations, and SQL tasks. The customer needs to provide job sequences and schedules as a prerequisite for converting them into Databricks workflows.

2 | DLT provides a standard framework for building batch and streaming use cases. It also includes critical data engineering features such as automatic data testing, deep pipeline monitoring and recovery. It also has out-of-the-box functionality for Slow Change Dimension (SCD) Type 1 and Type 2 tables.

3 | It is also possible to use external tools like Apache Airflow. Considering how tightly coupled Databricks Workflows is with the Databricks Intelligence Platform, it is recommended that Databricks Workflows be used for better integration, simplicity and lineage.

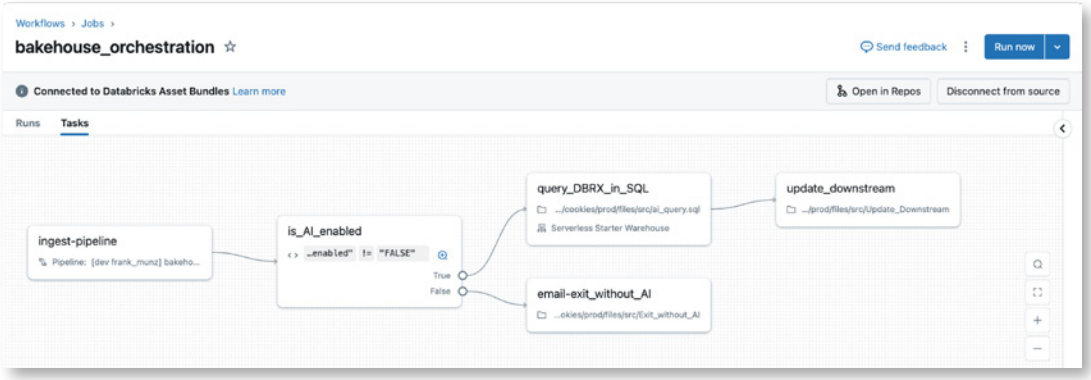


Figure 5: Databricks Workflows

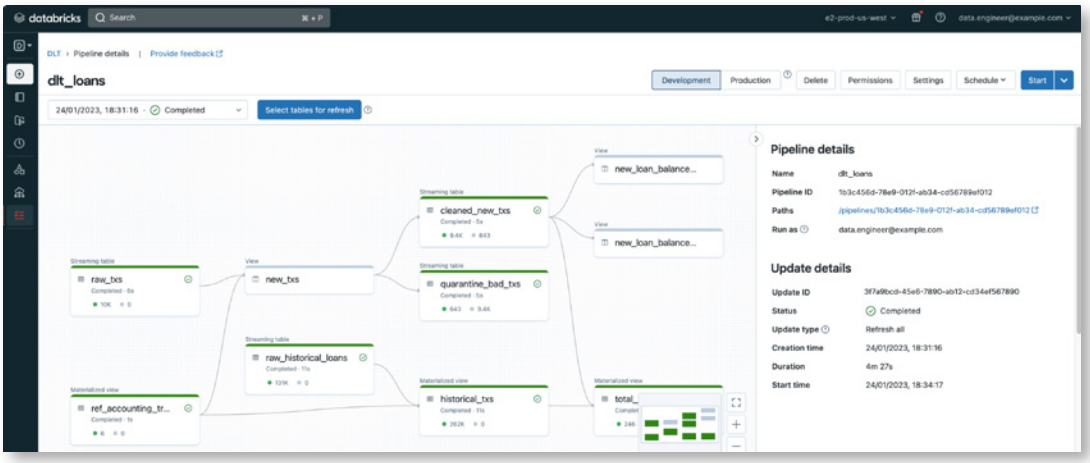


Figure 6: DLT pipelines

QUERY MIGRATION AND REFACTORING

Migrating from T-SQL to Databricks SQL requires identifying and replacing any incompatible/proprietary T-SQL functions or syntax. Databricks have mature code converters and migration tooling to make this process smoother and highly automated.

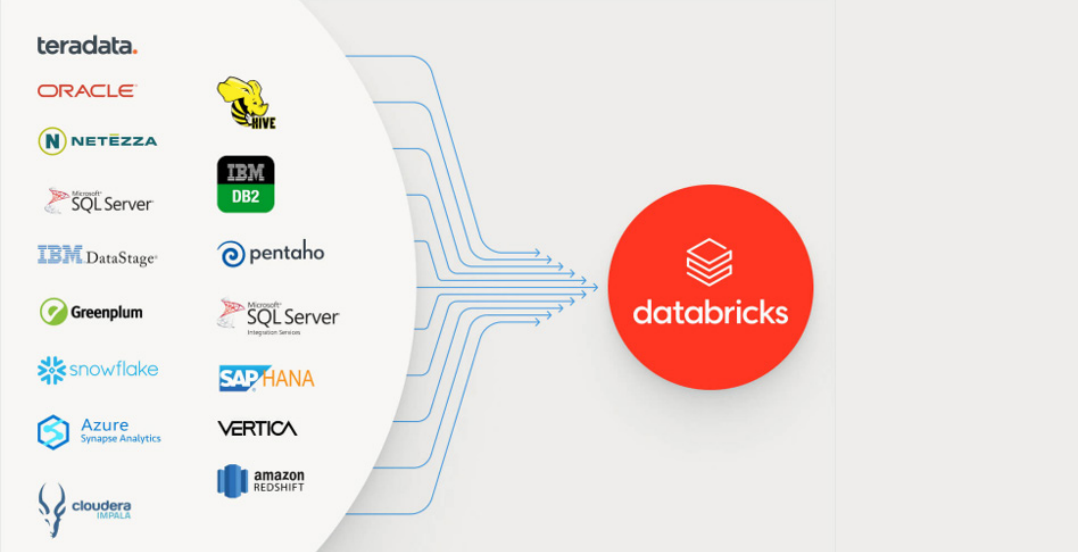


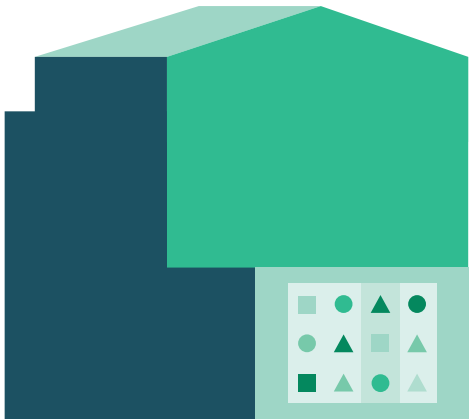
Figure 7: Databricks migration tooling simplifies legacy EDW migrations

Databricks Code Converter (acquired from BladeBridge) offers automated tooling to modernize and convert Microsoft SQL Server code to Databricks.

- **Automated conversion:** Databricks Converter can automatically convert SQL workloads, significantly speeding up and de-risking migration projects.
- **Broad support:** It supports a wide range of legacy EDW and ETL platform syntax and can convert legacy code to Databricks.
- **Broad adoption by services firms:** Most System Integrator partners have deep expertise and access to our converters.
- **Cost and time-effective:** Our Converter reduces the cost and time required for a migration project by automating the process.
- **Decreases complexity:** The tool reduces the complexity of the migration process by providing a systematic approach to conversion.

Since SQL Server and Databricks support ANSI SQL standards, many T-SQL queries can be automatically converted to Databricks syntax to accelerate the migration. The Databricks Code Converter supports schema conversion (tables and views), SQL queries (select statements, expressions, functions, user-defined functions, etc.), stored procedures and data loading utilities such as COPY INTO. The conversion configuration is externalized, meaning users can extend conversion rules during migration projects to handle new code pattern sets to achieve a more significant percentage of automation. A migration proposal with automated converter tooling can be created for your organization via Databricks Professional Services or our certified Migration Brickbuilder SI Partners. Databricks Code Converter tooling requires Databricks professional services or a Databricks SI Partner agreement.

Please review this short demonstration of the conversion tool.



Code Optimization

Many queries will likely need to be refactored and optimized during the migration process. Easy techniques like automated liquid clustering and predictive optimization make performance tuning almost an automated process in Databricks. Predictive Optimization uses techniques like:

1. Compaction - that optimizes file sizes.
2. Liquid clustering - that incrementally clusters incoming data, enabling optimal data layout and efficient data skipping.
3. Running Vacuum - which reduces costs by deleting unneeded files from storage.
4. Automatic updating of Statistics - running the ANALYZE STATISTIC command on the required columns for best performance.

Figure 8: Automatic liquid clustering

SQL Server to Databricks Cutover Phase

During this phase, while data workloads run concurrently in SQL Server and Databricks, it presents an opportunity for a comparative analysis to understand the behavior of workloads in Databricks versus SQL Server. This can help identify potential bottlenecks or shortcomings resulting from the code migration and refactoring phase.

To minimize expenses and disruption to business during this transition, consider the following recommendations:

1. Develop a cutover schedule that prioritizes the most resource-intensive workloads.
2. Establish clear criteria for approvals for cutover to Databricks.
3. Define appropriate criteria for retiring workloads in SQL Server based on the approvals from step 2.
4. Implement an effective communication strategy and conduct downstream user enablement sessions to minimize business disruption.
5. Implement a round-the-clock hypercare phase to support critical business-related workloads during the transition.

With this phase complete, ETL workloads are fully migrated and operational in Databricks, and the final layer data in SQL Server is synchronized with the Gold layer data in Databricks.

Phase 5: BI and Analytics Tools Integration

To further consolidate data platform infrastructure and maintain a single source of truth data, organizations have adopted Databricks SQL to meet their data warehousing needs and support downstream applications and business intelligence dashboards.

Once ingestion and transformation pipelines are migrated to the Databricks Data Intelligence Platform, it is critical to ensure the business continuity of downstream applications and data consumers. Databricks Data Intelligence Platform has validated large-scale BI integrations with many popular BI tools like Tableau, Power BI, Qlik, ThoughtSpot, Sigma, Looker and more. The expectation for a given set of dashboards or reports to work is to ensure all the upstream tables and views are migrated along with the associated pipelines and dependencies to conform to the existing data models and semantic layers in those tools.

As described in the blog (see section 3.5 Repointing BI workloads), one common way to repoint BI workloads after data migration is to test sample reports, rename existing tables' data source/table names, and point to the new ones.

Typically, if the schema of the tables and views post-migration hasn't changed, repointing is a straightforward exercise in handling switching databases on the BI dashboard tool. If the schema of the tables has changed, you will need to modify the tables/views in the lakehouse to match the expected schema of the report/dashboard and publish them as a new data source for the reports.

Many customers take this opportunity to optimize their BI models and semantic layers to align with business needs.

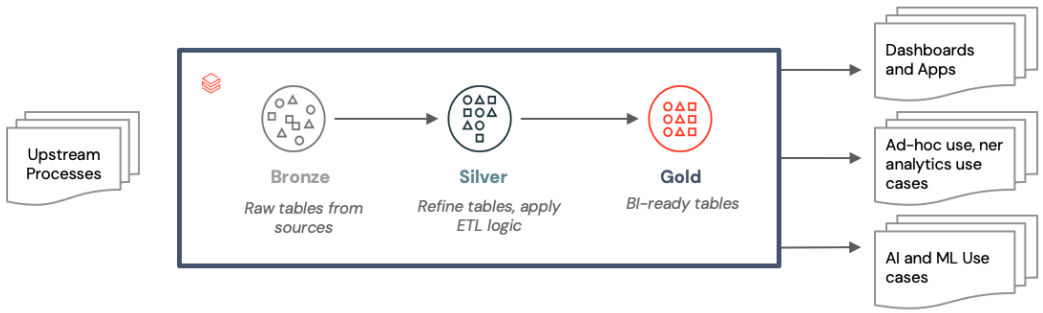
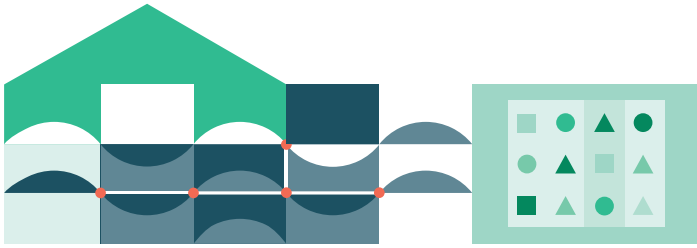


Figure 9: Future-state architecture

During report migration, you may encounter a scenario where expanding the permissions of BI tool access to cloud storage buckets becomes necessary to leverage Databricks Cloud Fetch feature. This feature enables high-bandwidth data exchange and enhances the efficiency of data retrieval. For more details, refer to the blog How We Achieved High-bandwidth Connectivity With BI Tools.



MICROSOFT POWER BI INTEGRATION

Microsoft Power BI, a commonly seen downstream application in various customer environments, typically operates on top of SQL Server's serving layer.

When migrating to Databricks Data Intelligence Platform, Power BI datasets must also be migrated. As Microsoft Power BI provides a connector for Azure Databricks, migrating Power BI datasets is a repointing to Databricks SQL warehouses. This can be performed by changing the data sources in M-code[1] using Power BI Desktop, SQL Server Management Studio or Tabular Editor.

When migrating Power BI datasets to Azure Databricks, standard migration techniques may apply, including piloting or MVP, using separate Power BI workspaces for testing and data validation after switching datasets to Azure Databricks SQL.

For more information on implementing Semantic Lakehouse with Azure Databricks and Power BI, please refer to the following blog posts:

- The Semantic Lakehouse With Azure Databricks and Power BI
- Power Up Your BI With Microsoft Power BI and Lakehouse in Azure Databricks: Part 1 — Essentials
- Power Up Your BI With Microsoft Power BI and Lakehouse in Azure Databricks: Part 2 — Tuning Power BI
- Power Up Your BI With Microsoft Power BI and Lakehouse in Azure Databricks: Part 3 — Tuning Azure Databricks SQL

1. A proprietary functional language primarily used for data transformation, allowing for the import, filtering, merging and shaping of data

Phase 6: Migration Validation

The primary validation method for a data pipeline is the resulting dataset itself. We recommend establishing an automated testing framework that can be applied to any pipeline. Typically, this involves using a testing framework with a script capable of automatically comparing values in both platforms.

Databricks recommends you perform the following checks at a minimum:

- Check to see if a table exists
- Check the counts of rows and columns across the tables
- Calculate various aggregates over columns and compare, for example:
  - SUM, MIN, MAX, AVG of numeric columns
  - MIN, MAX for string and date/time columns
  - COUNT(*), COUNT(NULL), COUNT(DISTINCT) for all columns

Run the pipelines in parallel for a specific period (we find one week to be an acceptable baseline, but you may wish to extend this to ensure stability) and review the comparison results to ensure the data is ingested and transformed into the proper context.

It is advisable to initiate validation with your most critical tables, which often drive the results or calculations of tables in the gold layer. This includes control tables, lookup tables and other essential datasets.

A robust data validation requires the following components:

- **Snapshot(s):** Data to work with, including a *pre-* and *post-version* for each script (ideal) and job being migrated.
- **Table comparison code:** A standardized way to compare the result table to determine whether the test is successful. The tables can be compared based on:
  - schema checks
  - row count checks
  - row-by-row checks

For more advanced table data and schema comparison, tools like Datacompy can be used.

Identifying the primary key combination from the customer is essential to check counts and row-by-row comparisons.

Need Help Migrating?

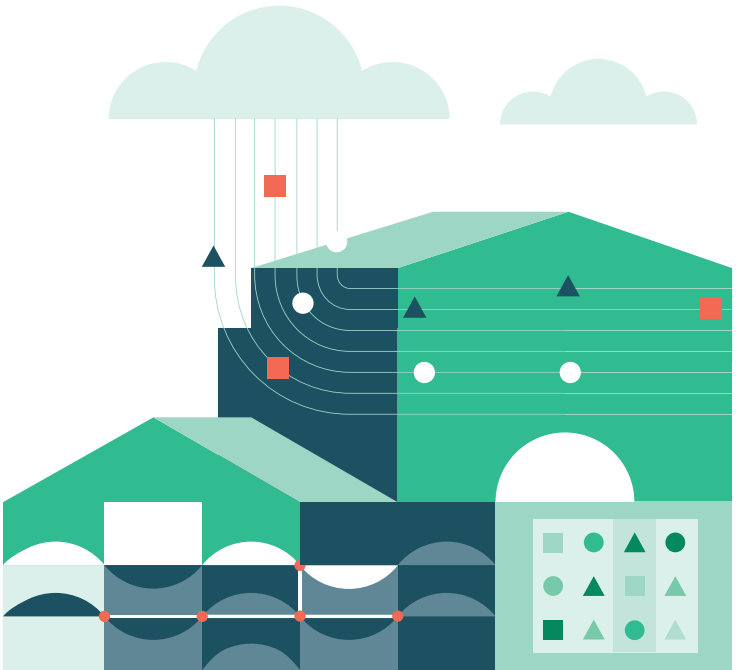
Regardless of size and complexity, the Databricks Professional Services team and an ecosystem of certified migration services partners and ISV partners offer different levels of support (advisory/assurance, staff augmentation, scoped implementation) to accelerate your migration and ensure successful implementation.

When engaging with our experts, you can expect:

- **Discovery and Profiling:** Our team starts by clearly understanding migration drivers and identifying challenges within the existing Microsoft SQL Server deployment. We conduct collaborative discussions with key stakeholders, leveraging automated profiling tools to analyze legacy workloads. This is used to determine drivers of business value and total cost of ownership (TCO) savings achievable with Databricks.
- **Assessment:** Using automated tooling, we perform an analysis of existing code complexity and architecture. This assessment helps estimate migration effort and costs, refine migration scope and determine which parts of the legacy environment require modernization or can be retired.
- **Migration Strategy and Design:** Our architects will work with your team to finalize the target Databricks architecture, detailed migration plan and technical approaches for the migration phases outlined in this guide. We will help select appropriate migration patterns, tools and delivery partners and collaborate with our certified SI partners to develop a comprehensive Statement of Work (SOW).

- **Execute and Scale:** We and our certified partners deliver on our comprehensive migration plan and then work with your team to facilitate knowledge sharing and collaboration and scale successful practices across the organization. Our experts can help you set up a Databricks Center of Excellence (CoE) to capture and disseminate lessons learned and drive standardization and best practices as you expand to new use cases.

Contact your Databricks representative or use this form for more information. Our specialists can help you every step of the way!



About Databricks

Databricks is the data and AI company. More than 10,000 organizations worldwide — including Block, Comcast, Condé Nast, Rivian, Shell and over 60% of the Fortune 500 — rely on the Databricks Data Intelligence Platform to take control of their data and put it to work with AI. Databricks is headquartered in San Francisco, with offices around the globe, and was founded by the original creators of Lakehouse, Apache Spark™, Delta Lake and MLflow.

To learn more, follow Databricks on LinkedIn, X and Facebook.

Start your free trial

© Databricks 2025. All rights reserved. Apache, Apache Spark, Spark and the Spark logo are trademarks of the Apache Software Foundation. Privacy Notice | Terms of Use