

Databricks Certified Data Engineer Associate

Ultimate Preparation Course + Practice Exam





Ramesh Retnasamy
Data Engineer/ Machine Learning Engineer



LinkedIn

<https://www.linkedin.com/in/ramesh-retnasamy/>

INSTRUCTOR

Ramesh Retnasamy

Senior Data Engineer/ Machine Learning Engineer

Udemy Instructor Partner

Total students Reviews

181,141 **39,927**



Azure Databricks & Spark For Data Engineers:Hands-on Project

Ramesh Retnasamy

4.6 ★★★★★ (20,616)

20 total hours · 190 lectures · All Levels

Bestseller



Azure Data Factory For Data Engineers - Project on Covid19

Ramesh Retnasamy

4.5 ★★★★★ (15,081)

13 total hours · 146 lectures · All Levels

Bestseller



Azure Synapse Analytics For Data Engineers - Hands On Project

Ramesh Retnasamy

4.6 ★★★★★ (3,521)

13.5 total hours · 132 lectures · All Levels

Bestseller



Comprehensive SQL Course For Data Professionals [NEW]

Ramesh Retnasamy

4.7 ★★★★★ (420)

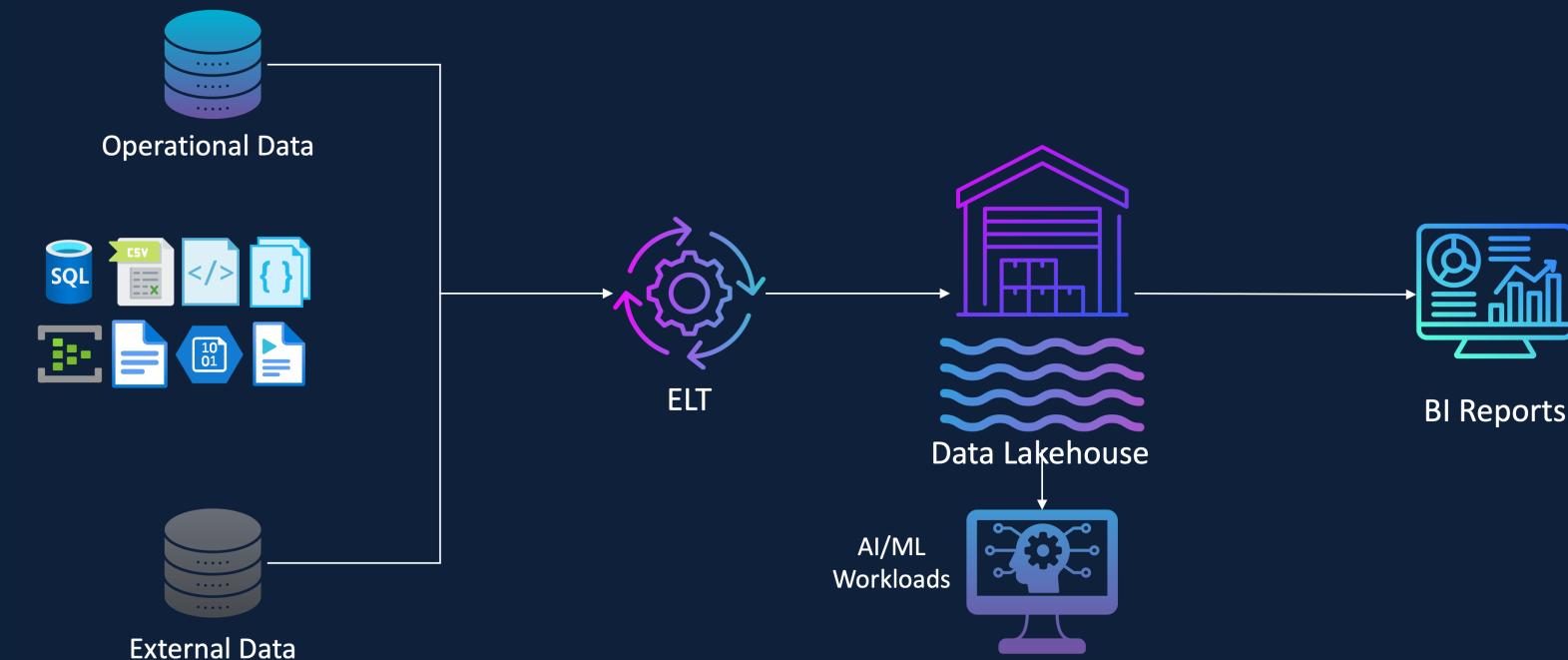
13.5 total hours · 130 lectures · All Levels

Databricks Certified Data Engineer Associate

Ultimate Preparation Course + Practice Exam



What is Data Lakehouse?



Handles All Types of Data

Cheap Cloud Object Storage

Uses Open File Format

Support for All Workloads

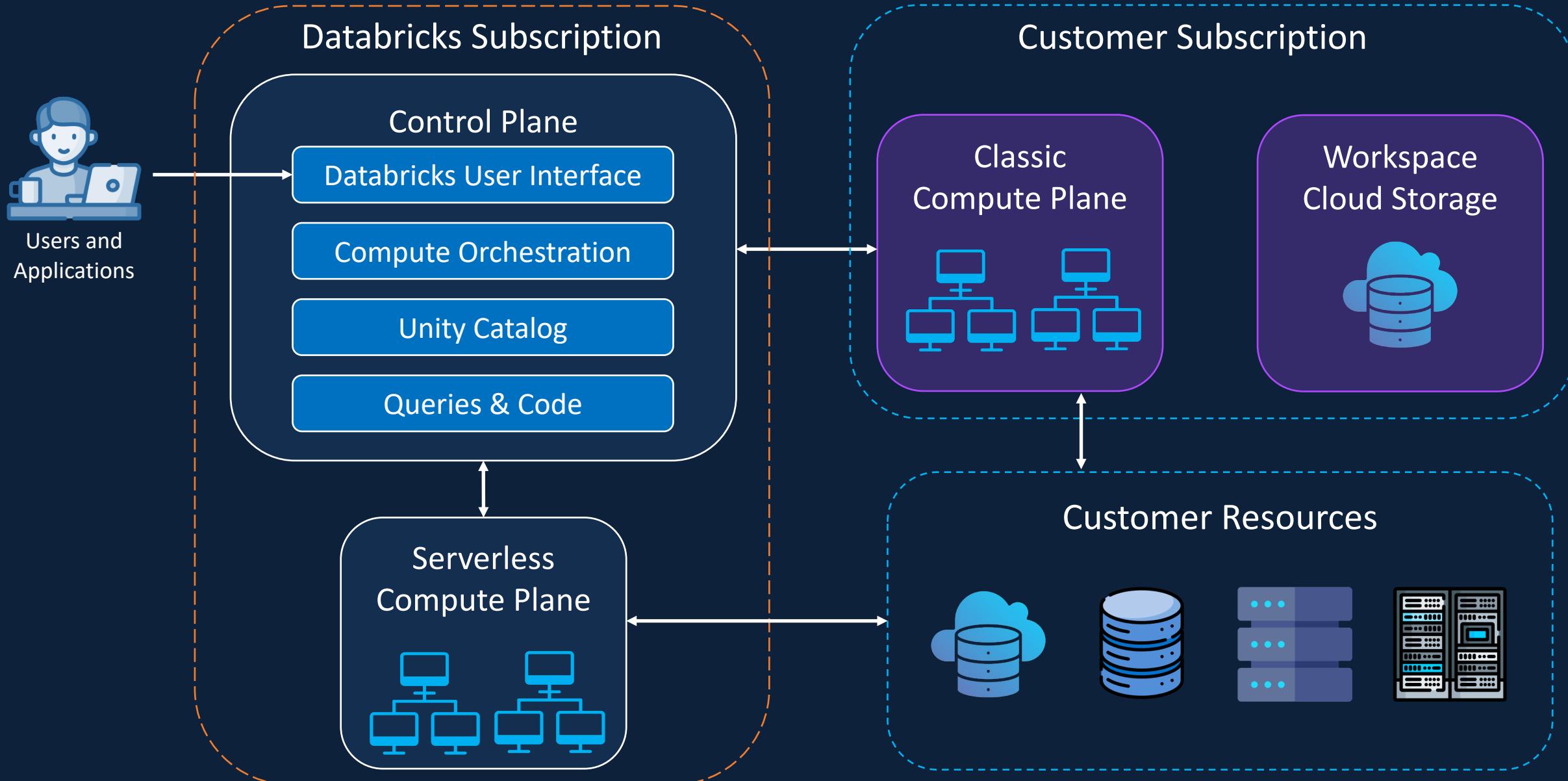
Direct BI Integration

ACID Support & Version History

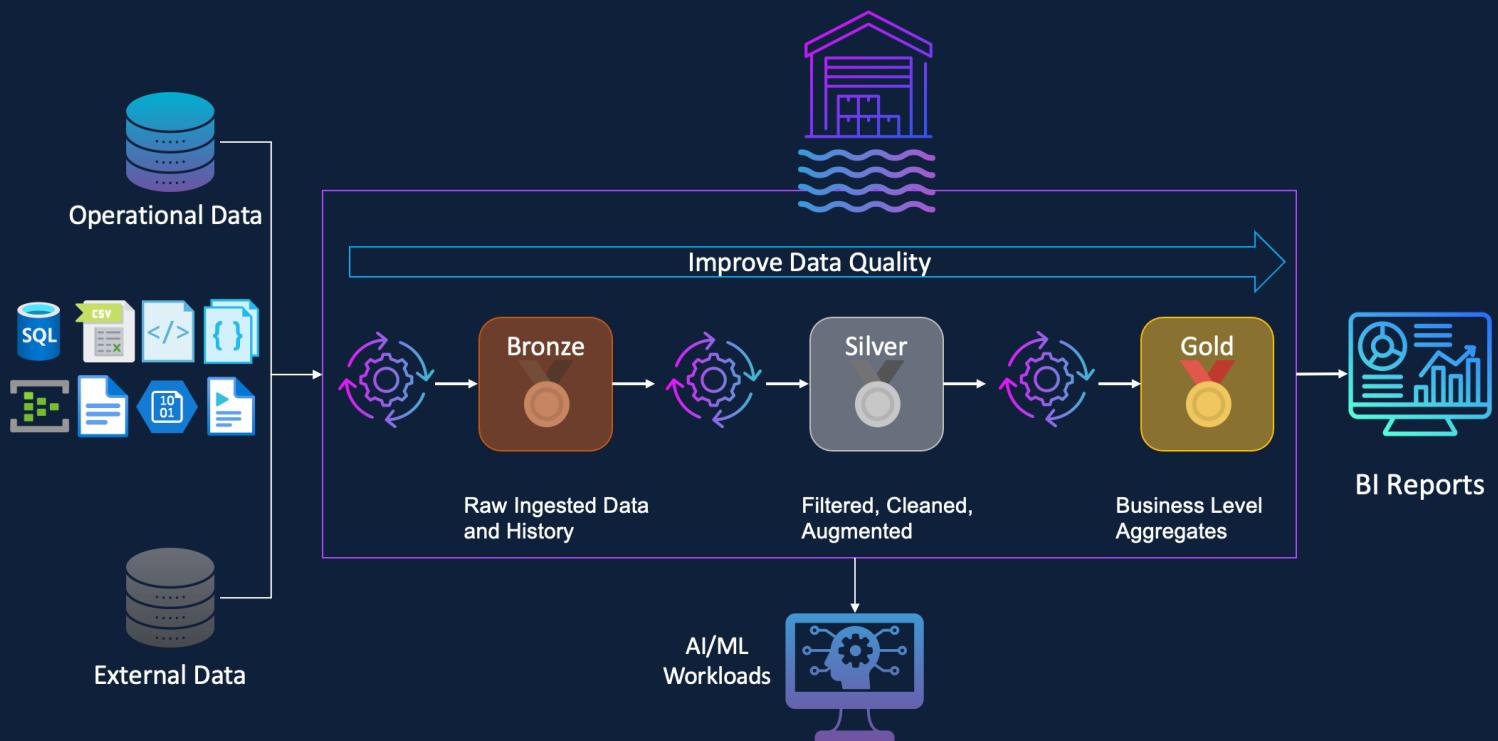
Improved Performance

Simple Architecture

Databricks Architecture



Medallion Architecture - Benefits



Data Lineage & Traceability

Data Governance & Compliance

Support for Incremental Processing

Enhanced Performance & Scalability

Data Security

Delta Live Tables (DLT) Architecture

Delta Live Tables (DLT)

User Interface

DLT Notebooks

SQL

Python



Data Transformations

Data Quality Expectations

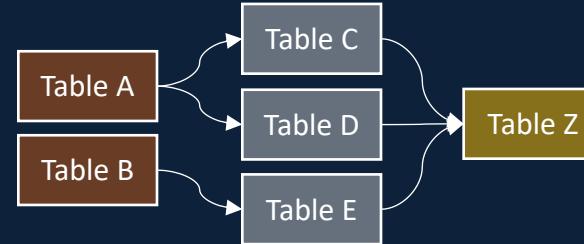
DLT Pipelines



Hive Metastore

Unity Catalog

DAG



Infrastructure



Metadata

Pipeline Status/ Health

Data Quality Checks

Data Lineage

Data Quality Checks

Dependency Management

Automated Checkpointing & Re-tries

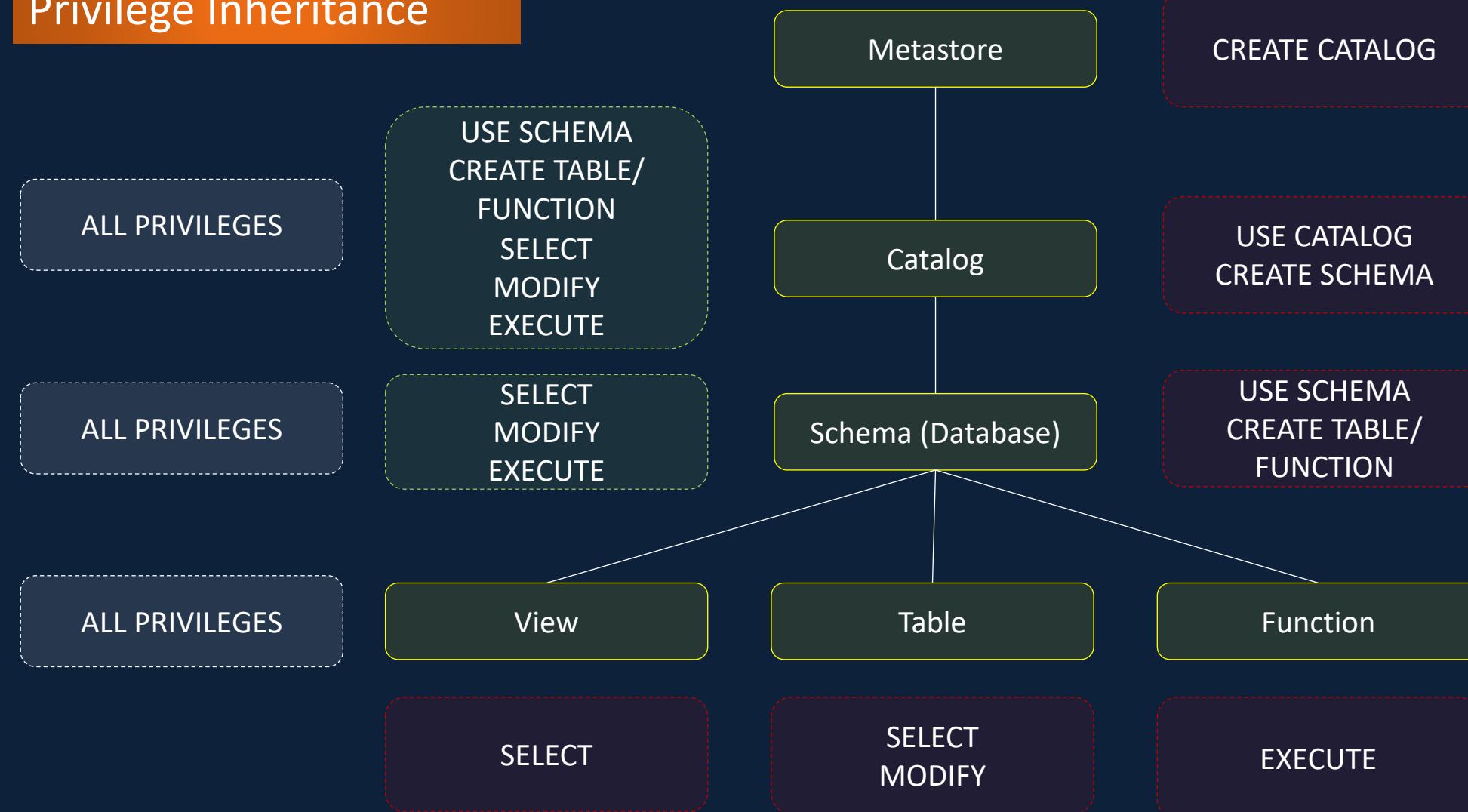
Incremental Processing

Infrastructure Management

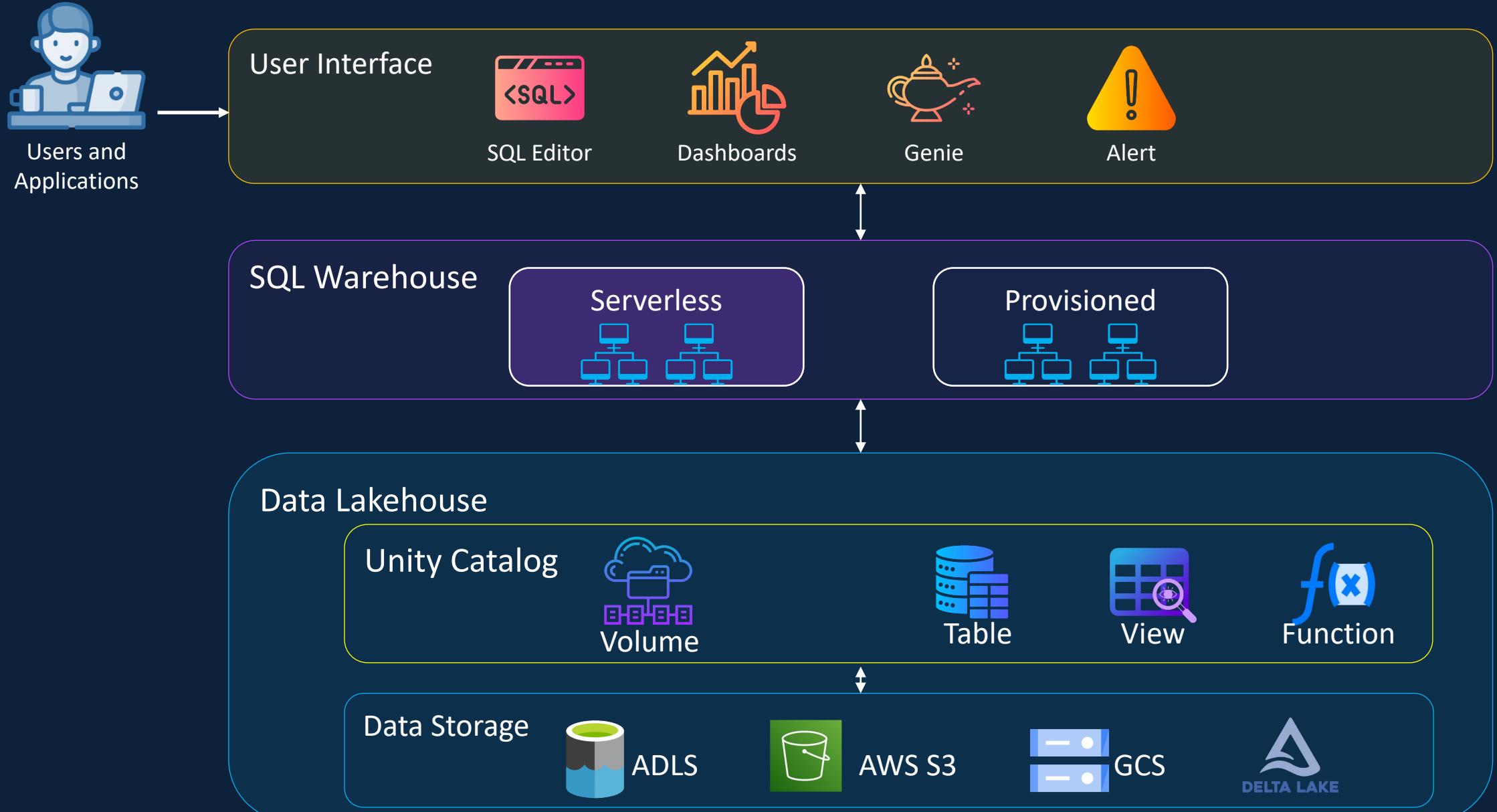
Data Lineage

Unity Catalog - Security Model

Privilege Inheritance

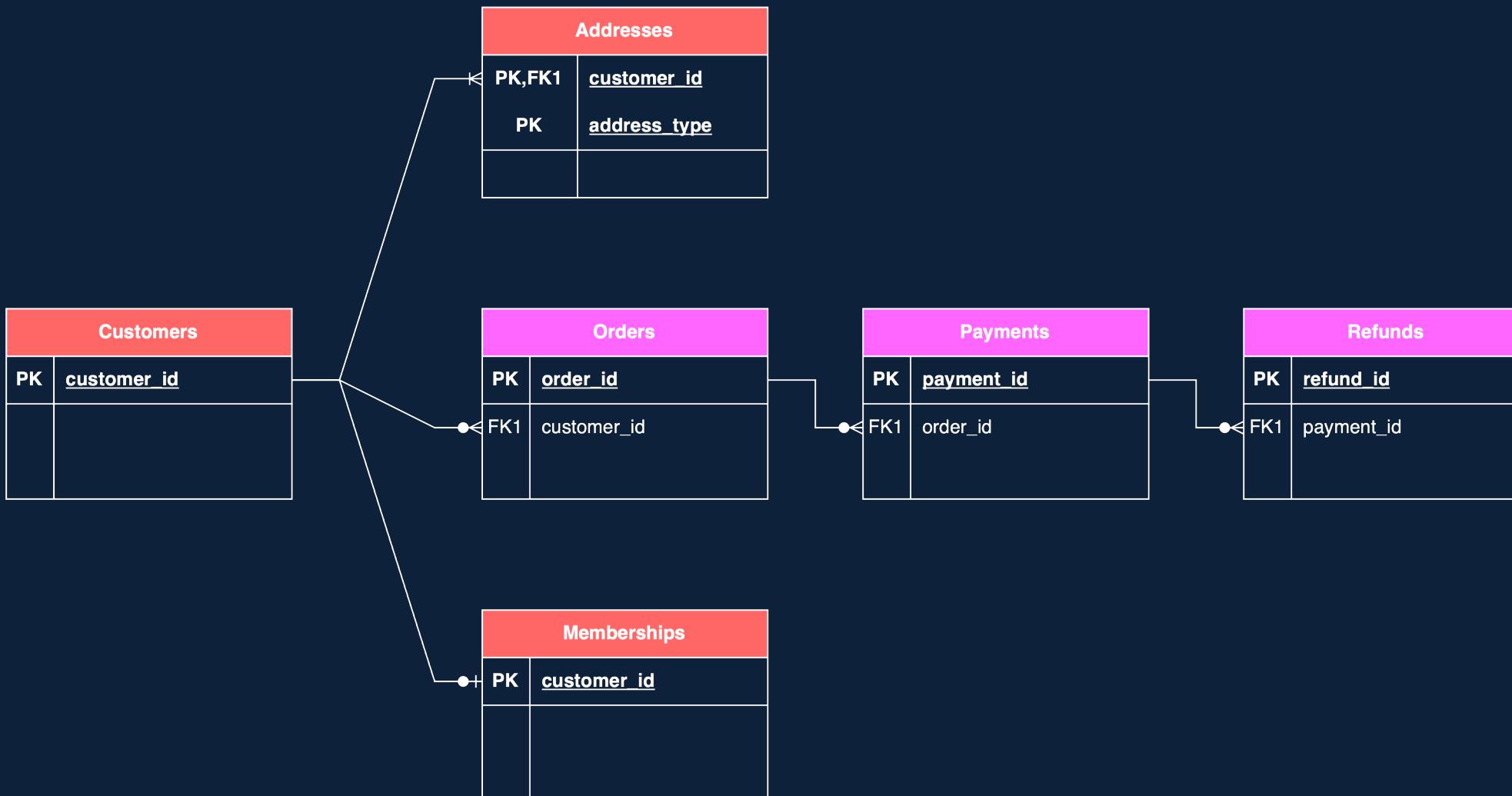


Databricks SQL



Tell me and I forget,
Teach me and I may remember,
Involve me and I learn!
- Benjamin Franklin

Data Lakehouse Project - GizmoBox



Workspace

- [dea03-etl-with-apache-s...](#)
- [01. Set-up Project Environment](#)
- [02. Extract Customers Data - JSON](#)
- [03. Extract Orders Data - Comple...](#)
- [04. Extract Memberships Data - I...](#)
- [05. Extract Addresses Data - CSV...](#)
- [06. Extract Payments Data - CSV ...](#)
- [07. Extract Refunds Data - via JD...](#)
- [08. Extract Data using Python](#)
- [09. Data Profiling](#)
- [10. Transform Customers Data](#)
- [11. Transform Payments Data](#)
- [12. Transform Refunds Data](#)
- [13. Transform Memberships Data](#)
- [14. Transform Addresses Data](#)
- [15. Query Orders Data - JSON Str...](#)
- [16. Transform Orders Data - Strin...](#)
- [17. Transform Orders Data - Explor...](#)
- [18. Join Customer Address](#)
- [19. Monthly Order Summary](#)
- [20. User Defined Functions](#)

Transform Addresses Data

1. Create one record for each customer with 2 sets of address columns, 1 for shipping and 1 for billing address
2. Write transformed data to the Silver schema

Dec 04, 2024 (3s)

2

SQL

```
SELECT customer_id,  
       address_type,  
       address_line_1,  
       city,  
       state,  
       postcode  
  FROM gizmo_box.bronze.v_addresses;
```

Table

	customer_id	address_type	address_line_1	city	state	postcode	_rescue
1		9179	shipping	9713 Medina Mountains Apt. 813	Lake Erika	Mississippi	91945
2		9179	billing	084 Anne Hollow Apt. 064	East Jasontown	Minnesota	77329
3		5816	shipping	8400 Rebecca Lodge Suite 011	Lake Lisatown	Montana	11354
4		5816	billing	869 Linda Locks Apt. 105	West Melissabury	Michigan	37662
5		4858	shipping	4838 Jacob Neck	Jamesview	Wisconsin	58657
6							

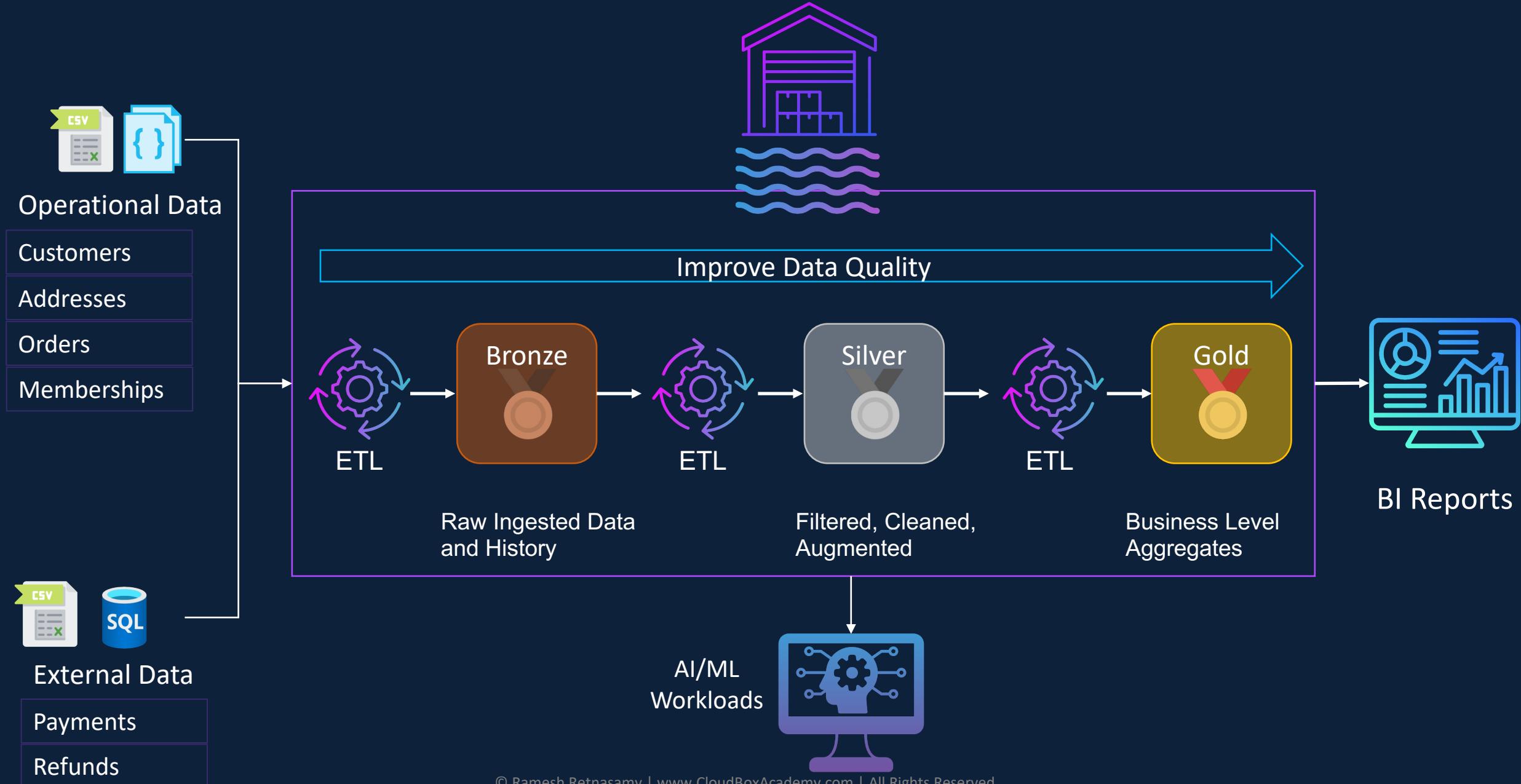
↓ 100 rows | 3.42s runtime

Refreshed 85 days ago

1. Create one record for each customer with both addresses, one for each address_type

[Documentation for PIVOT clause](#)

Data Architecture - GizmoBox



19. Monthly Order Summary

SQL



File Edit View Run Help Last edit was 59 days ago

▶ Run all

Unknown

Schedule

Share

Workspace



← dea03-etl-with-apache-s...



01. Set-up Project Environment

02. Extract Customers Data - JSON

03. Extract Orders Data - Comple...

04. Extract Memberships Data - I...

05. Extract Addresses Data - CSV...

06. Extract Payments Data - CSV ...

07. Extract Refunds Data - via JD...

08. Extract Data using Python

09. Data Profiling

10. Transform Customers Data

11. Transform Payments Data

12. Transform Refunds Data

13. Transform Memberships Data

14. Transform Addresses Data

15. Query Orders Data - JSON Str...

16. Transform Orders Data - Strin...

17. Transform Orders Data - Explor...

18. Join Customer Address

19. Monthly Order Summary

20. User Defined Functions

Monthly Order Summary

Markdown



For each of the customer, produce the following summary per month

1. total orders
2. total items bought
3. total amount spent

▶ Dec 30, 2024 (1s)

2

SQL



```
SELECT * FROM gizmobox.silver.orders WHERE customer_id = 5816 ;
```

▶ (2) Spark Jobs

Table +

	order_id	order_status	payment_method	total_amount	transaction_timestamp	customer_id	ite
1	12	Cancelled	PayPal	999	2024-12-06 20:26:49	5816	
2	10	Pending	Credit Card	399	2024-10-07 22:09:27	5816	
3	11	Pending	Bank Transfer	897	2024-10-13 17:34:19	5816	

↓ 3 rows | 0.67s runtime

Refreshed 59 days ago

▶ Dec 30, 2024 (7s)

3

```
CREATE TABLE IF NOT EXISTS gizmobox.gold.order_summary_monthly
AS
SELECT date_format(transaction_timestamp, 'yyyy-MM') AS order_month,
       customer_id,
       COUNT(DISTINCT order_id) AS total_orders,
```

Delta Live Tables (DLT) Project - CircuitBox



06/02/2025, 12:53:44 · Completed

Select tables for refresh ⚡

ⓘ

Graph

List



Streaming table

bronze_addresses

Completed · 2s

● - ● -

Streaming table

silver_addresses_...

Completed · 1s

● - ● -

Streaming table

silver_addresses

Completed · 1s

● - ● -

Streaming table

bronze_customers

Completed · 2s

● - ● -

Streaming table

silver_customers_...

Completed · 1s

● - ● -

Streaming table

silver_customers

Completed · 1s

● - ● -

Streaming table

bronze_orders

Completed · 5s

● 4 ● 0

Streaming table

silver_orders_clean

Completed · 3s

● 4 ● 0

Streaming table

silver_orders

Completed · 3s

● 4 ● 0

Materialized view

gold_customer_or...

Completed · 4s

● 13 ● 0



Full Length Practice Exam

Exam mode



Practice Exam 1 - Databricks Certified Data Engineer Associate

45 questions | 1 hour 30 minutes | 70% correct required to pass

This exam consists of 45 questions across the 5 domains of the Databricks Certified Data Engineer Associate Certification Exam. You will have 90 minutes to complete this exam.

Passing Score for the actual exam is 70%. If you are scoring 85% or higher on these practice exams, you are ready to take and pass the real Databricks Certified Data Engineer Associate Certification Exam!

Good luck!

Instructions:

- You can pause the test at any time and resume later.
- You can retake the test as many times as you would like.
- The progress bar at the top of the screen will show your progress as well as the time remaining. If you run out of time, don't worry; you will still be able to finish the test.
- You can click the bookmark icon to mark a question for review or click "skip question" to skip it entirely.
- Click "finish test" to finish the test and see your results immediately.

Exam mode ⓘ

1/45

⌚

1:29:21



Finish test

Question 1:

A **financial services company** processes large volumes of **transactional data** for fraud detection and compliance reporting.

- They need to run **scheduled ETL jobs** to clean, transform, and load data at regular intervals.
- The workloads are **not interactive** and do not require manual intervention.
- Cost efficiency is a priority, as clusters should automatically **terminate when the job is completed**.

Which type of **Databricks cluster** is best suited for this use case?

All-Purpose Cluster

Delta Cluster

High-Concurrency Cluster

NoSQL Cluster

Job Cluster

Exam mode ⓘ

33/45

⌚ 1:16:48 ||

Finish test

Question 33:

A **data analyst** has written the following SQL query to analyze sales data in Databricks:

```
1 | SELECT customer_id, SUM(total_amount) AS total_spent
2 | FROM sales_data
3 | GROUP BY customer_id
4 | ORDER BY total_spent DESC;
```

A **data engineer** wants to execute this SQL query inside a **PySpark notebook** while working with a **Spark DataFrame**.

Which of the following methods correctly allows the engineer to run this SQL command in PySpark?



```
1 | df = sql("""
2 |   SELECT customer_id, SUM(total_amount) AS total_spent
3 |   FROM sales_data
4 |   GROUP BY customer_id
5 |   ORDER BY total_spent DESC
6 |   """)
```



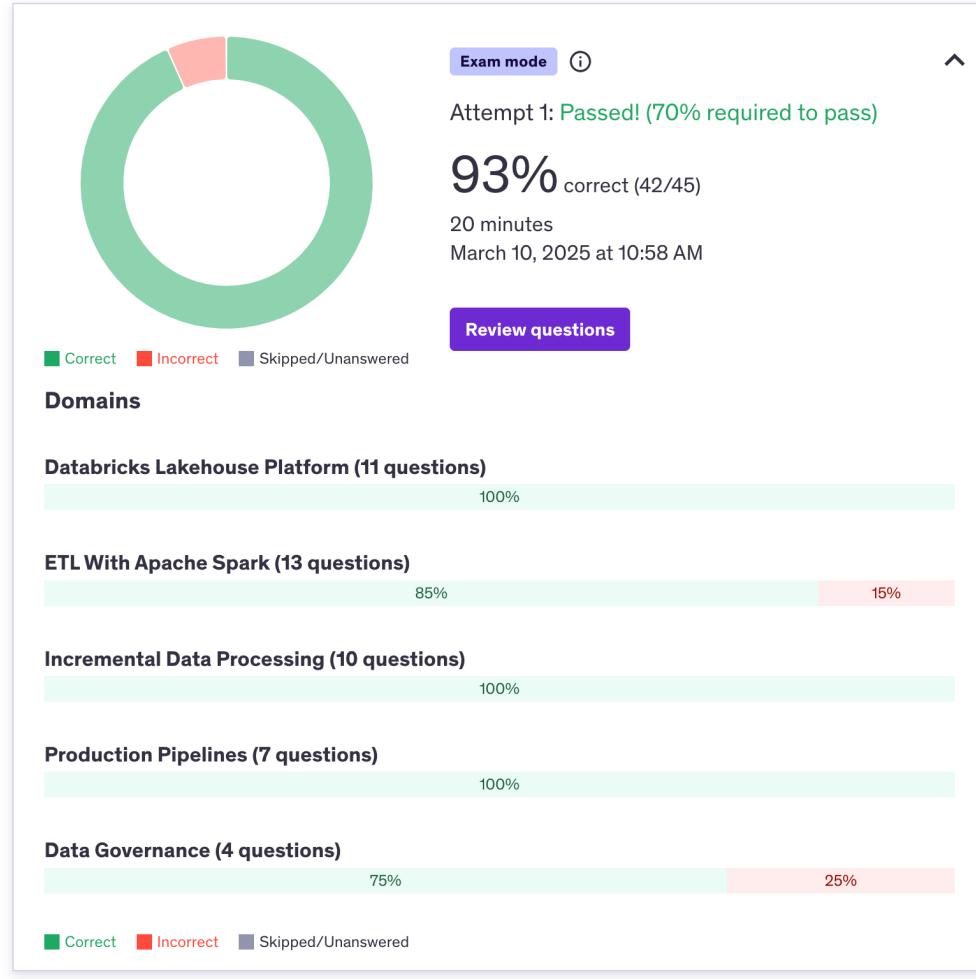
```
1 | df = spark.sql("""
2 |   SELECT customer_id, SUM(total_amount) AS total_spent
3 |   FROM sales_data
4 |   GROUP BY customer_id
5 |   ORDER BY total_spent DESC
6 |   """)
```



```
1 | df = spark.execute_sql("""
2 |   SELECT customer_id, SUM(total_amount) AS total_spent
```

Practice Exam 1 - Databricks Certified Data Engineer Associate - Results

45 questions | 1 hour 30 minutes | 70% correct required to pass





No Prior Databricks Knowledge Required

Basic SQL & Python Knowledge Required

Azure Cloud Subscription Required



Ask Questions, I will answer 😊

Udemy life time access

Udemy 30 day money back guarantee



Industry Recognition

Career Advancement

Skill Validation

Future-proof Your Career



ENROLL NOW!

DISCLAIMER

This course is independently created and is **not affiliated with or endorsed by Databricks, Inc.**

All content, explanations, and practice materials are original and created only for educational purposes.

This course doesn't contain actual exam questions.

All trademarks and product names are the property of their respective owners.

For official certification materials, please visit databricks.com.



Official Exam Guide - <https://www.databricks.com/learn/certification/data-engineer-associate>



Databricks Documentation- <https://docs.databricks.com/>



Course Structure

Course Structure

Exam Topics



Databricks Lakehouse Platform

ETL With Apache Spark

Incremental Data Processing

Production Pipelines

Data Governance



Official Exam Guide - <https://www.databricks.com/learn/certification/data-engineer-associate>

Course Structure

Pre-Requisite

2. Azure Subscription

Databricks Lakehouse Platform

3. Introduction to Databricks Lakehouse Platform

4. Databricks Workspace Components

5. Introduction to Unity Catalog

ETL With Apache Spark

6. Overview

7. Querying Data

8. Transforming Data

Incremental Data Processing

9. Spark Structured Streaming

10. Delta Lake

11. DLT - Overview

12. DLT – Pipelines and Notebooks

Production Pipelines

13. Databricks Jobs

Data Governance

14. Databricks SQL

15. Data Governance

Practice

16. Practice Exam



Creating Azure Free Subscription



Azure Portal Overview



What is Data Lakehouse?

What is Data Lakehouse?



A Data Lakehouse is a new, open data management architecture that combines the flexibility, cost-efficiency, and scale of Data Lakes with the data management and ACID transactions of Data Warehouses, enabling business intelligence (BI) and machine learning (ML) on all data.

- *Databricks*

What is Data Lakehouse?



A Data Lakehouse is a new, open data management architecture that combines the flexibility, cost-efficiency, and scale of Data Lakes with the data management and ACID transactions of Data Warehouses, enabling business intelligence (BI) and machine learning (ML) on all data.

What is Data Lakehouse?



Data Lakehouse

=



Data Lake

+



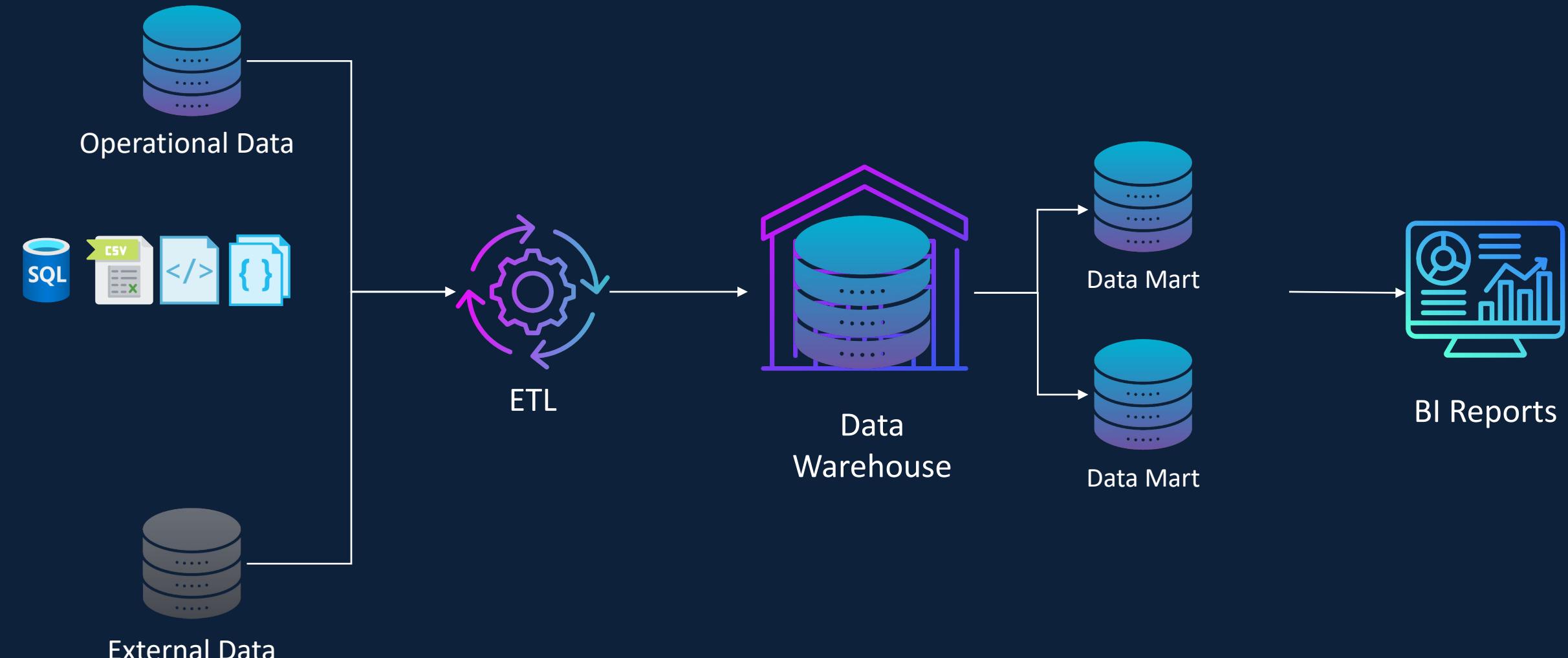
Data Warehouse

A Data Lakehouse is a new, open data management architecture that combines the flexibility, cost-efficiency, and scale of Data Lakes with the data management and ACID transactions of Data Warehouses, enabling business intelligence (BI) and machine learning (ML) on all data.

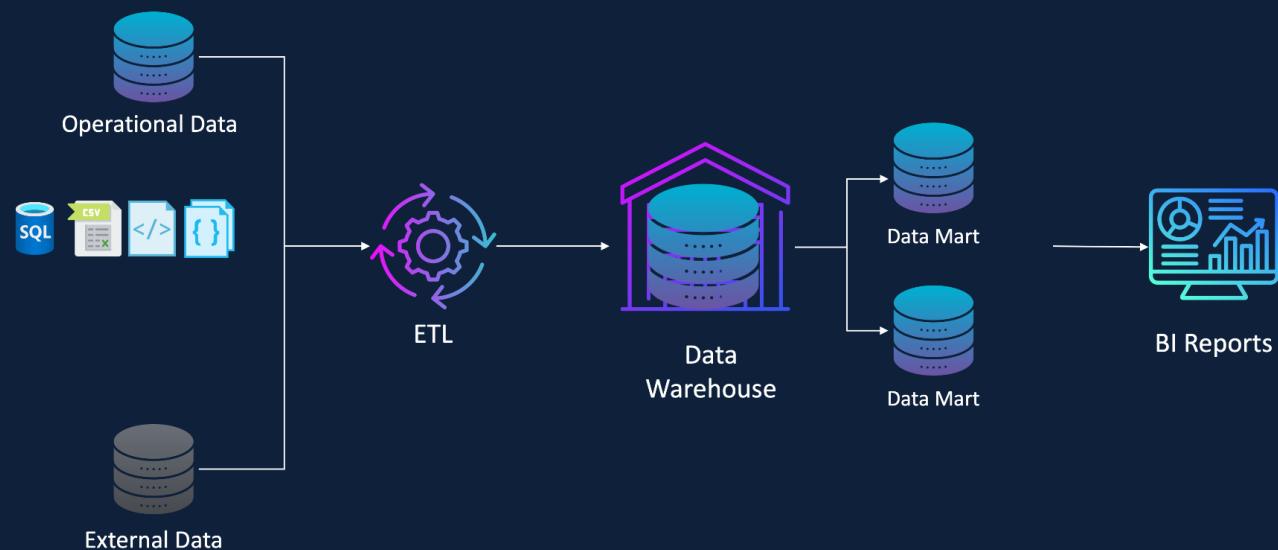


What is Data Warehouse?

What is Data Warehouse?



What is Data Warehouse?



Increased Data Volume & Variety

Longer Time to Ingest New Data

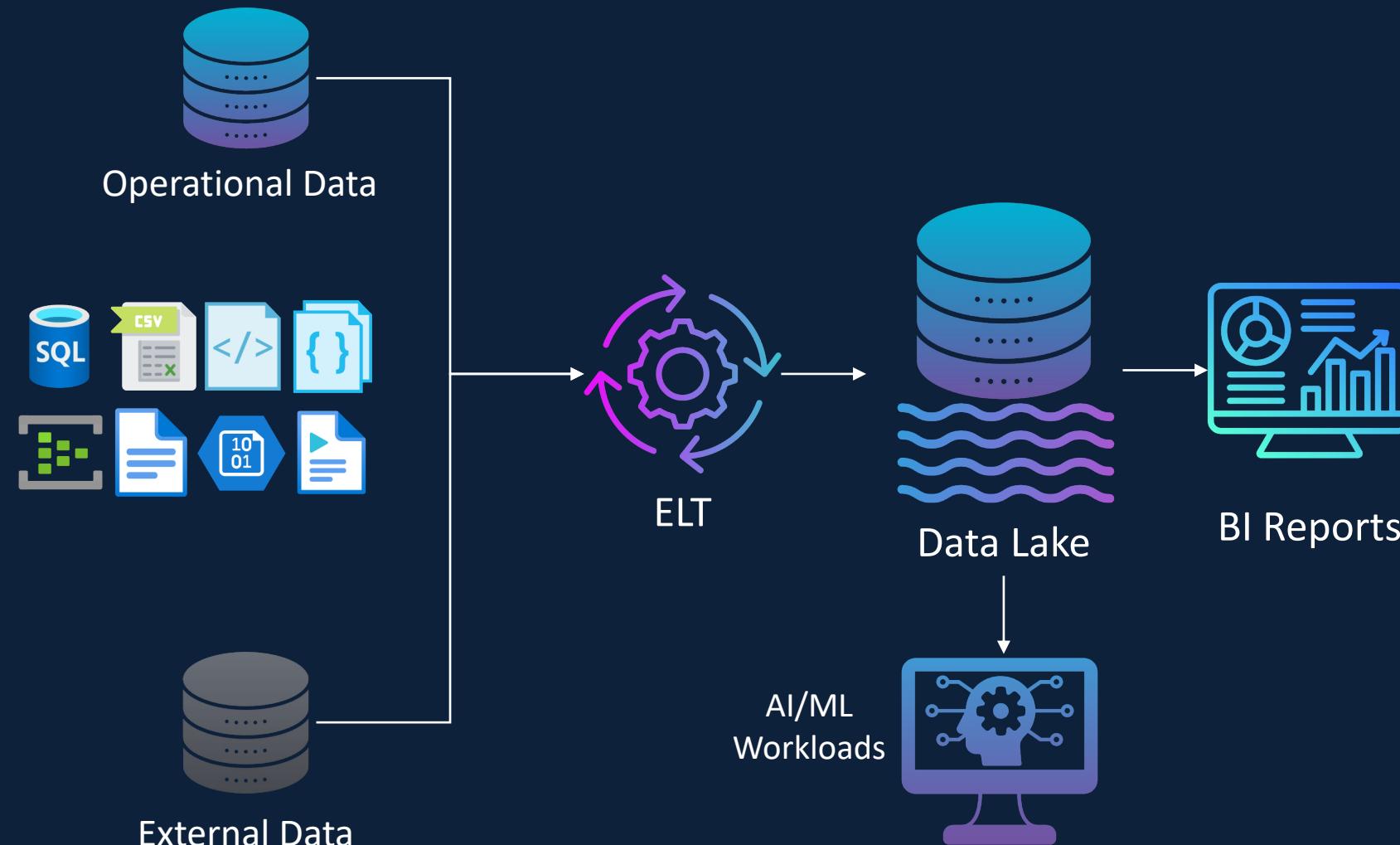
Proprietary Data Formats /Vendor Lock-in

Scalability Issues

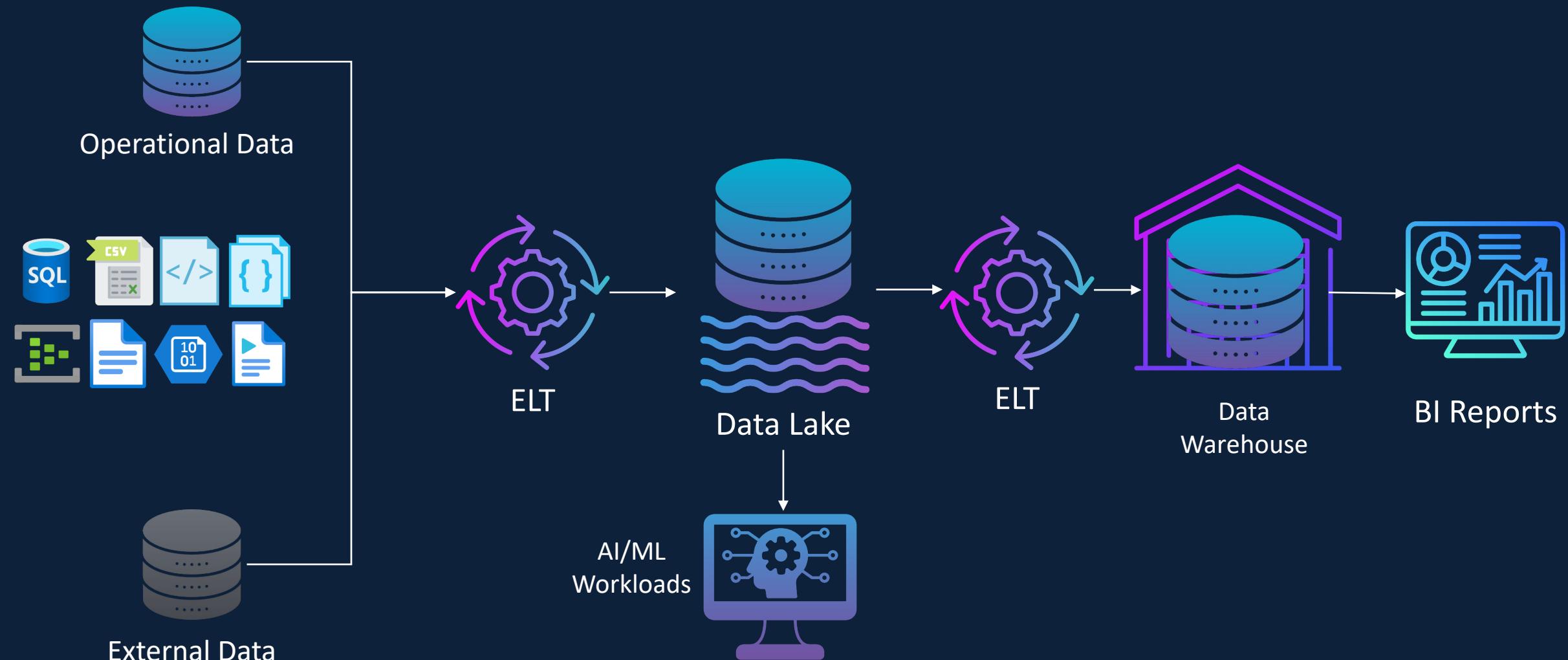
High Storage Costs

Limited Support for Advanced Analytics

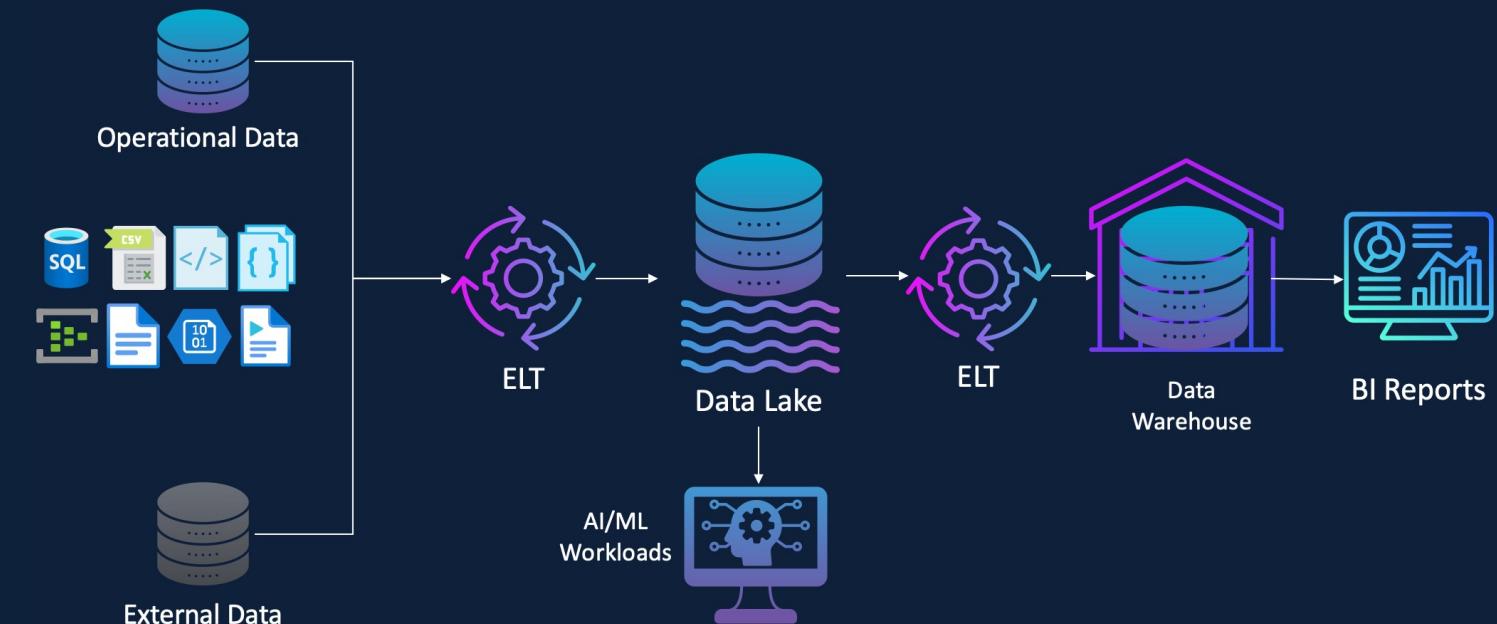
What is Data Lake?



What is Data Lake?



What is Data Lake?



No Support for ACID Transactions

Partial Data Loads

Inconsistent Reads

Complicated Data Corrections

No Rollback Capability

GDPR Challenges

No Data Versioning

Poor BI Performance and Support

Complex to set-up

Streaming vs. Batch Processing

What is Data Lakehouse?



Data Lake



Data Ware**house**

What is Data Lakehouse?



Data Lake



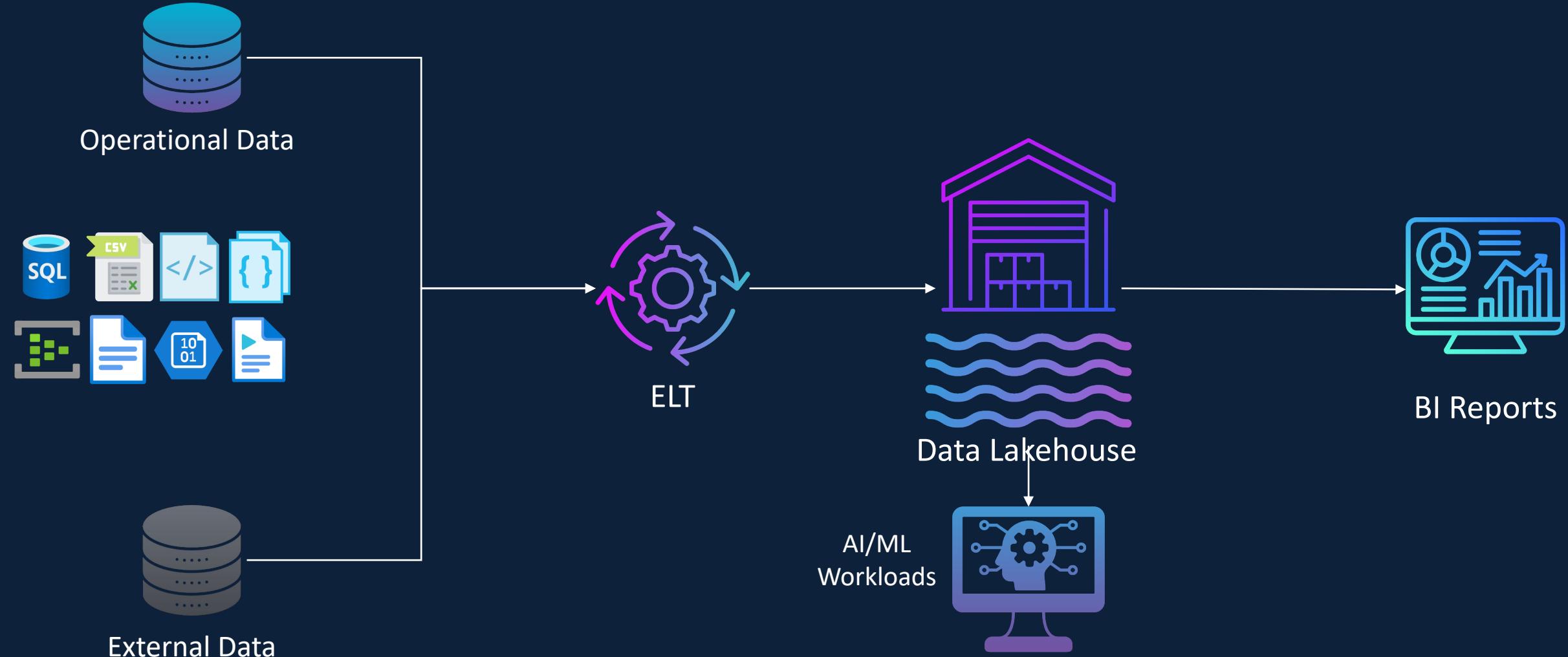
Data Warehouse



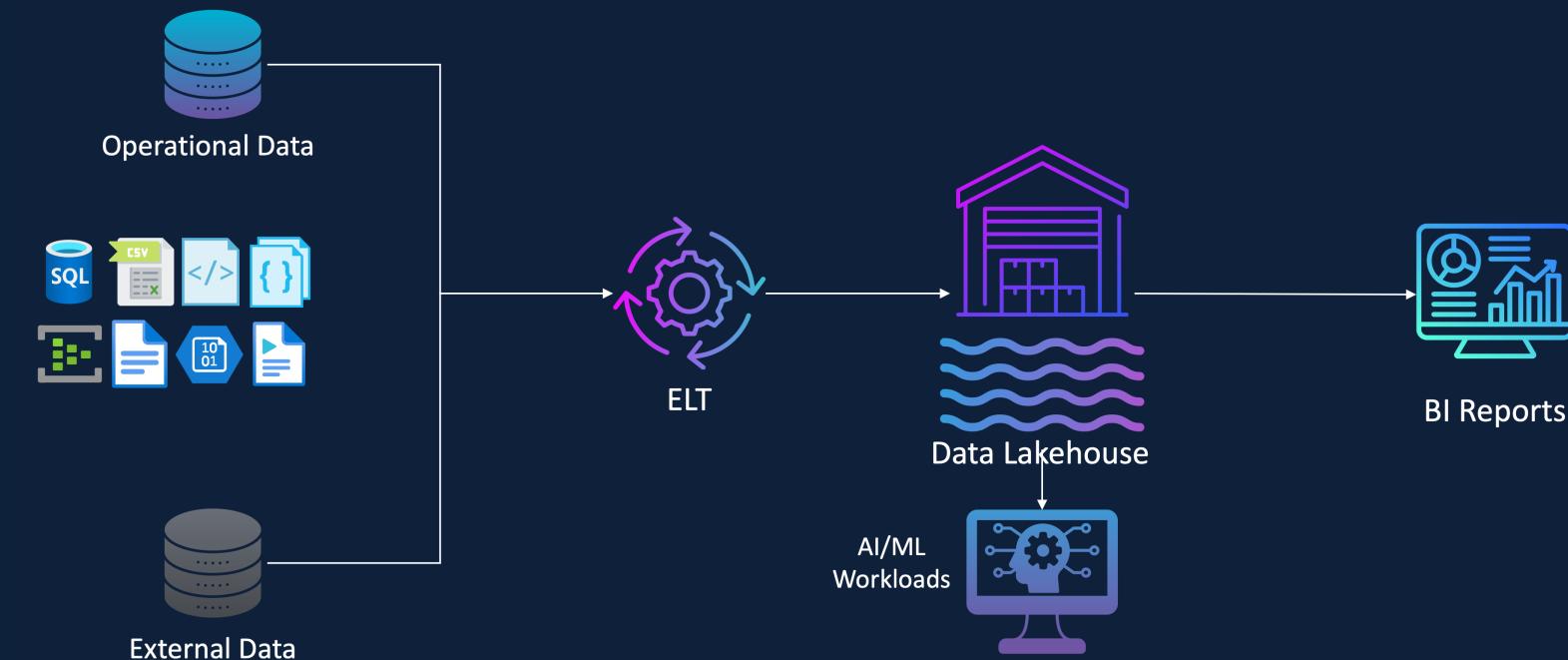
Data Lakehouse



What is Data Lakehouse?



What is Data Lakehouse?



Handles All Types of Data

Cheap Cloud Object Storage

Uses Open File Format

Support for All Workloads

Direct BI Integration

ACID Support & Version History

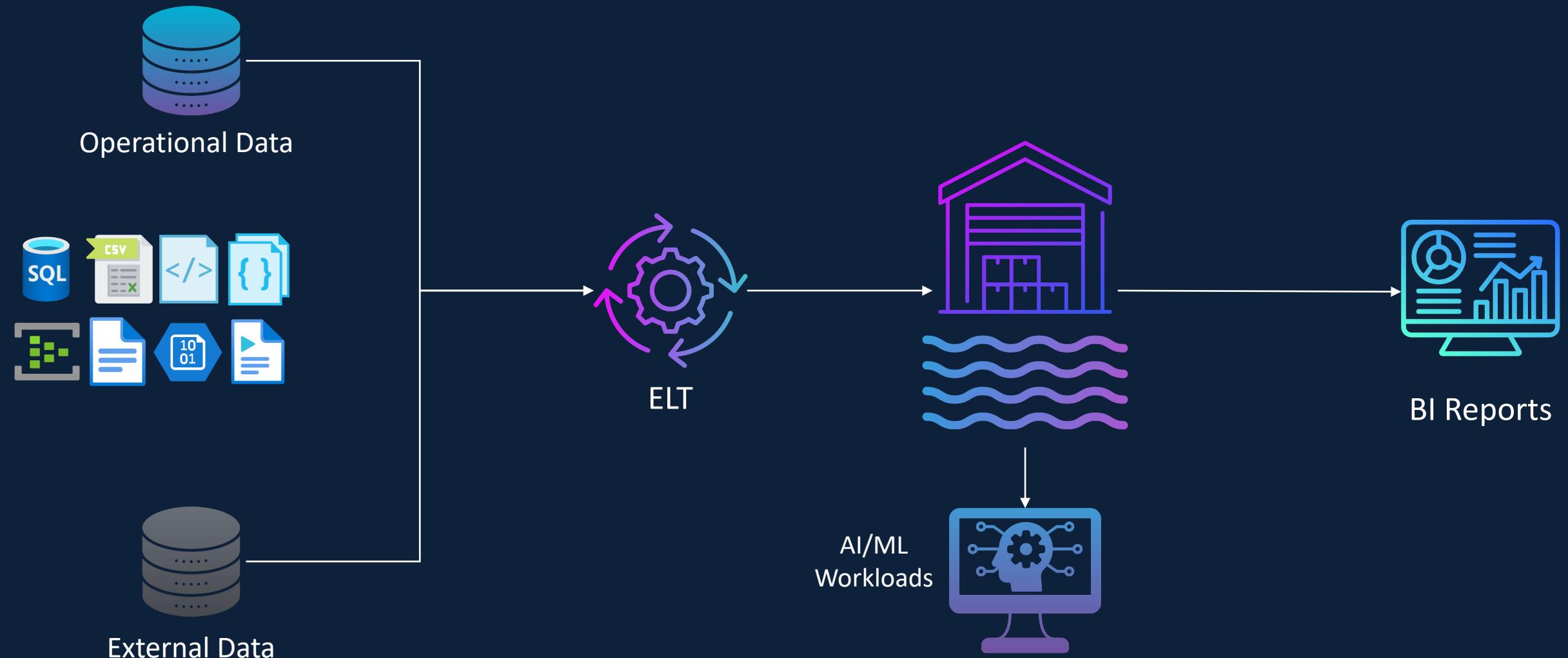
Improved Performance

Simple Architecture

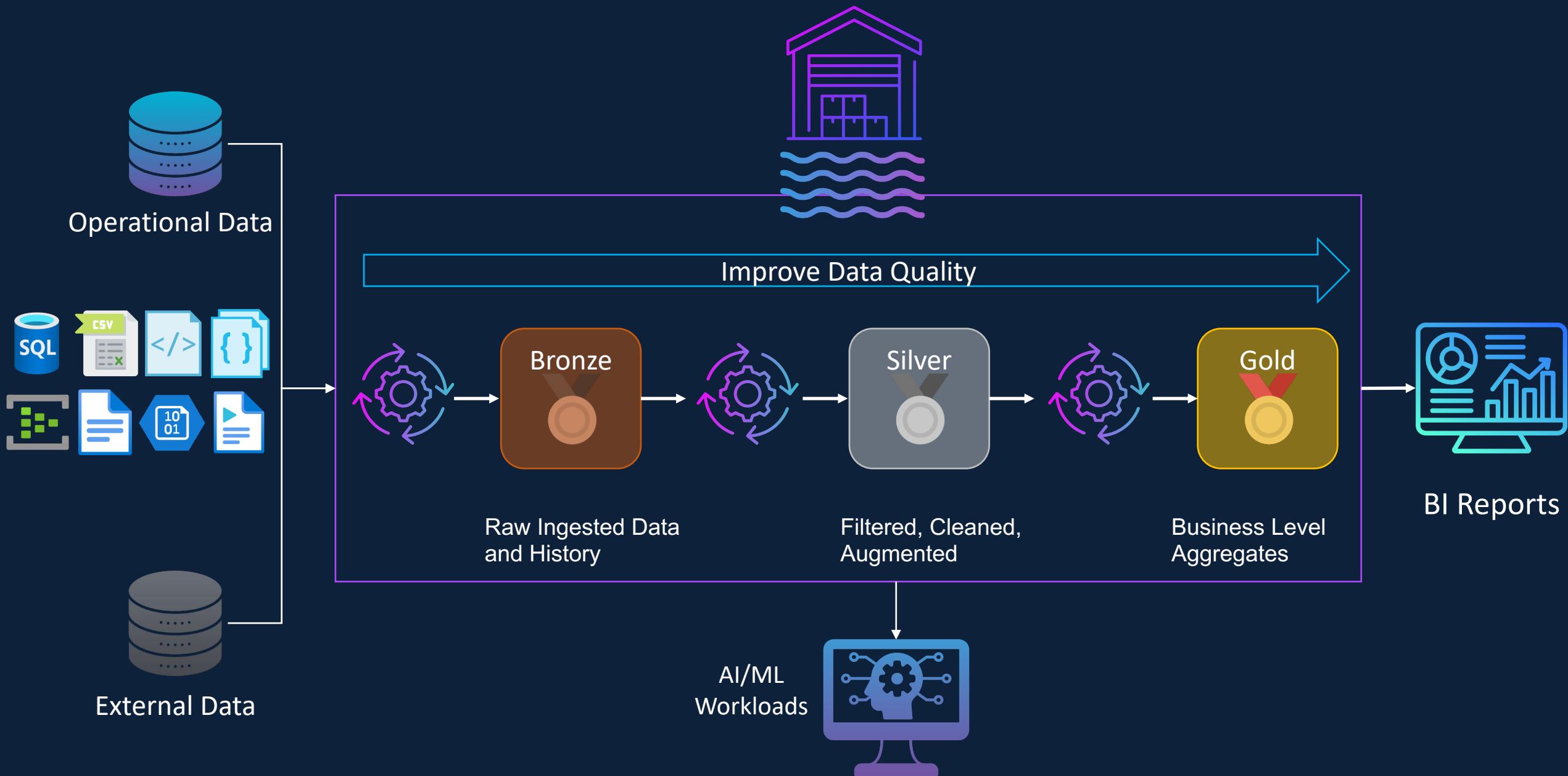


What is Medallion Architecture?

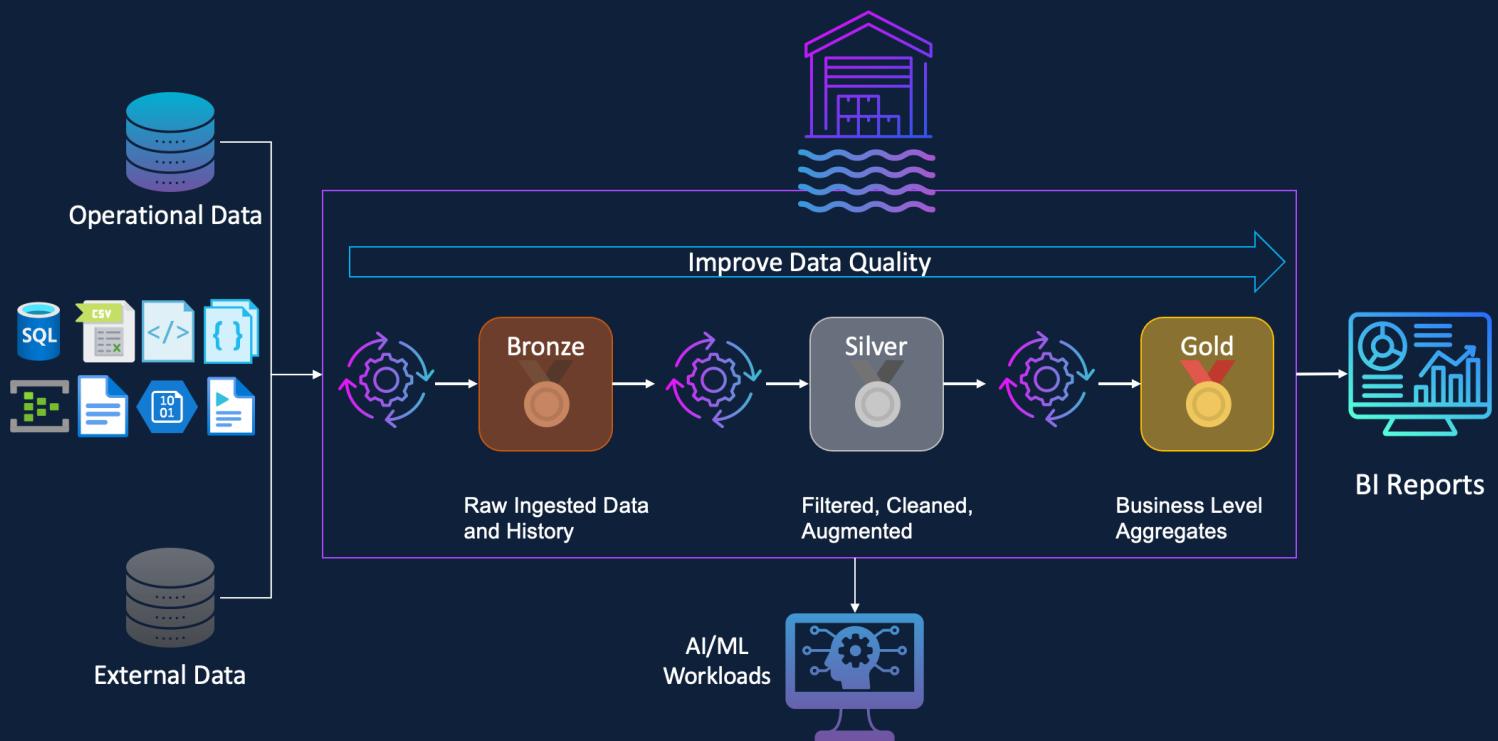
Medallion Architecture



Medallion Architecture



Medallion Architecture - Benefits



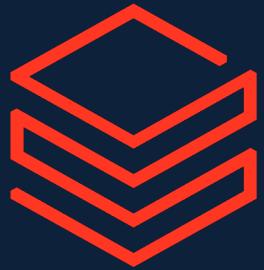
Data Lineage & Traceability

Data Governance & Compliance

Support for Incremental Processing

Enhanced Performance & Scalability

Data Security



Introduction to Databricks

Introduction to Databricks



Apache Spark

Apache Spark is a lightning-fast unified analytics engine for big data processing and machine learning



100% Open source under Apache License

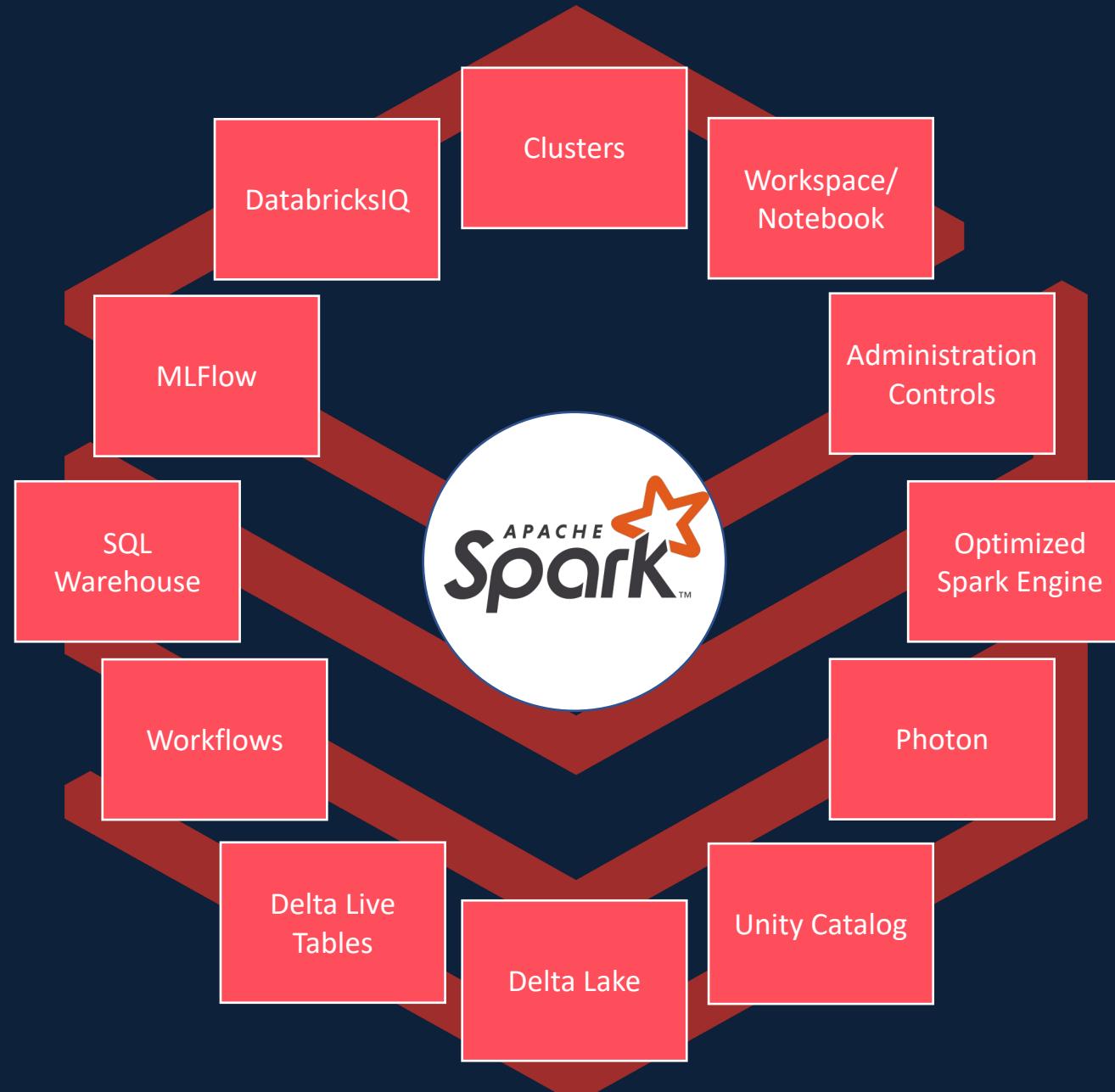
Simple and easy to use APIs

In-memory processing engine

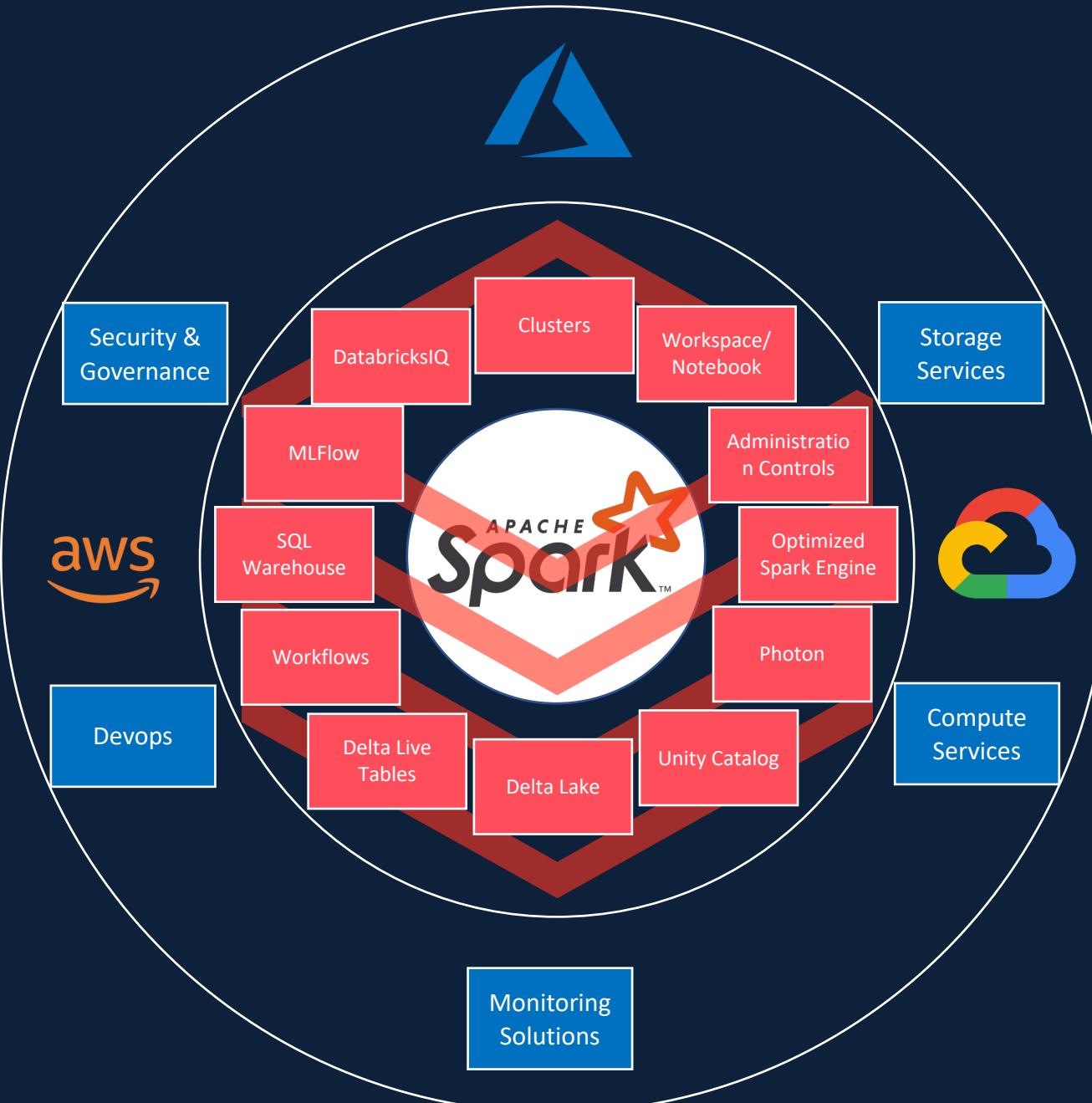
Distributed computing Platform

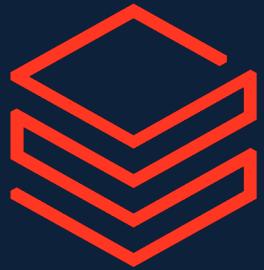
Unified engine which supports SQL, streaming, ML and graph processing

Databricks

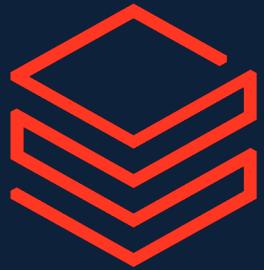


Databricks

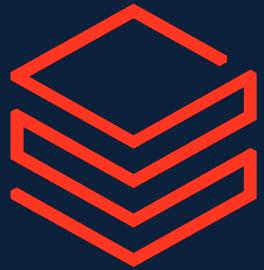




Creating a Databricks Service

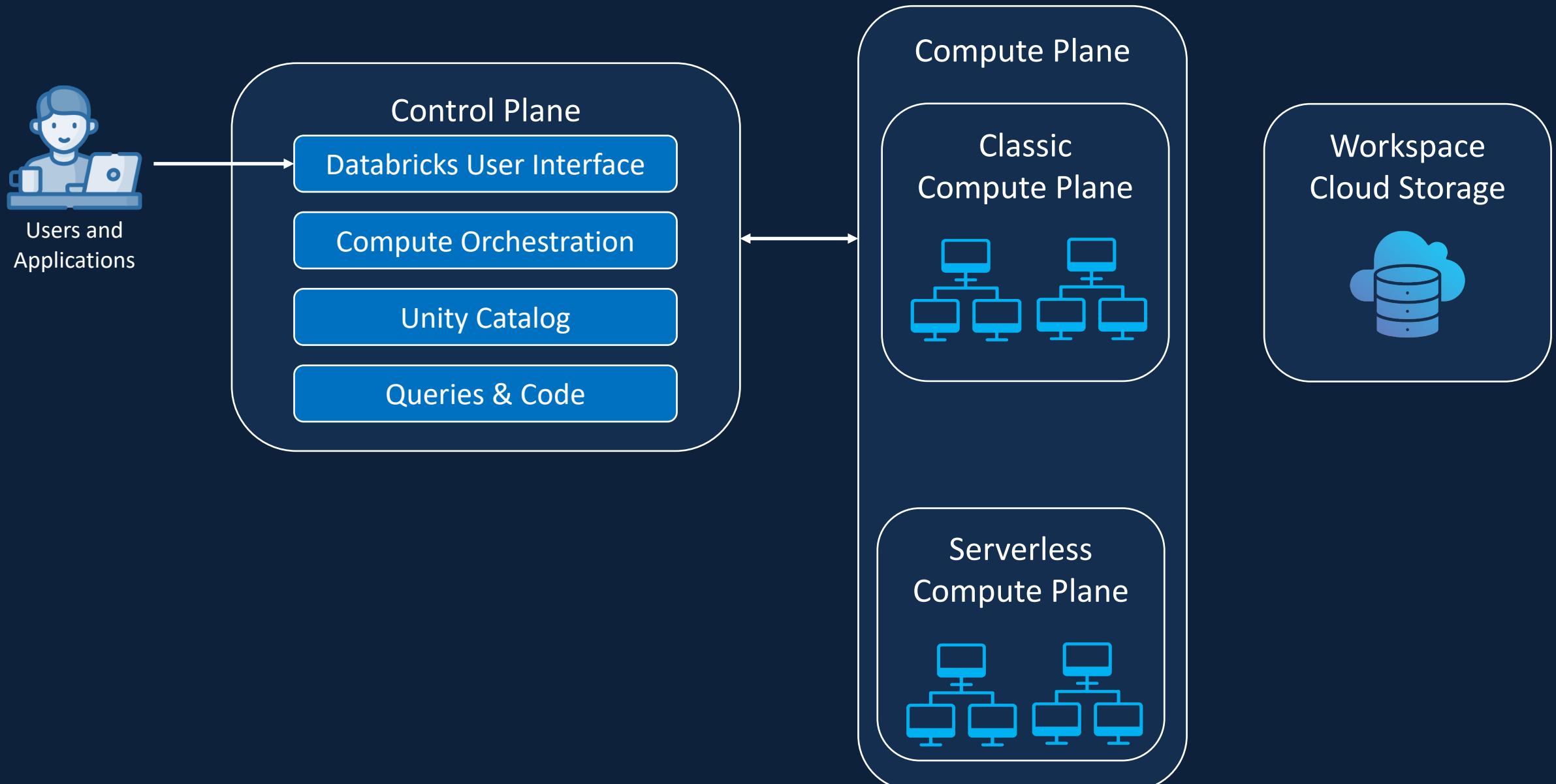


Databricks User Interface

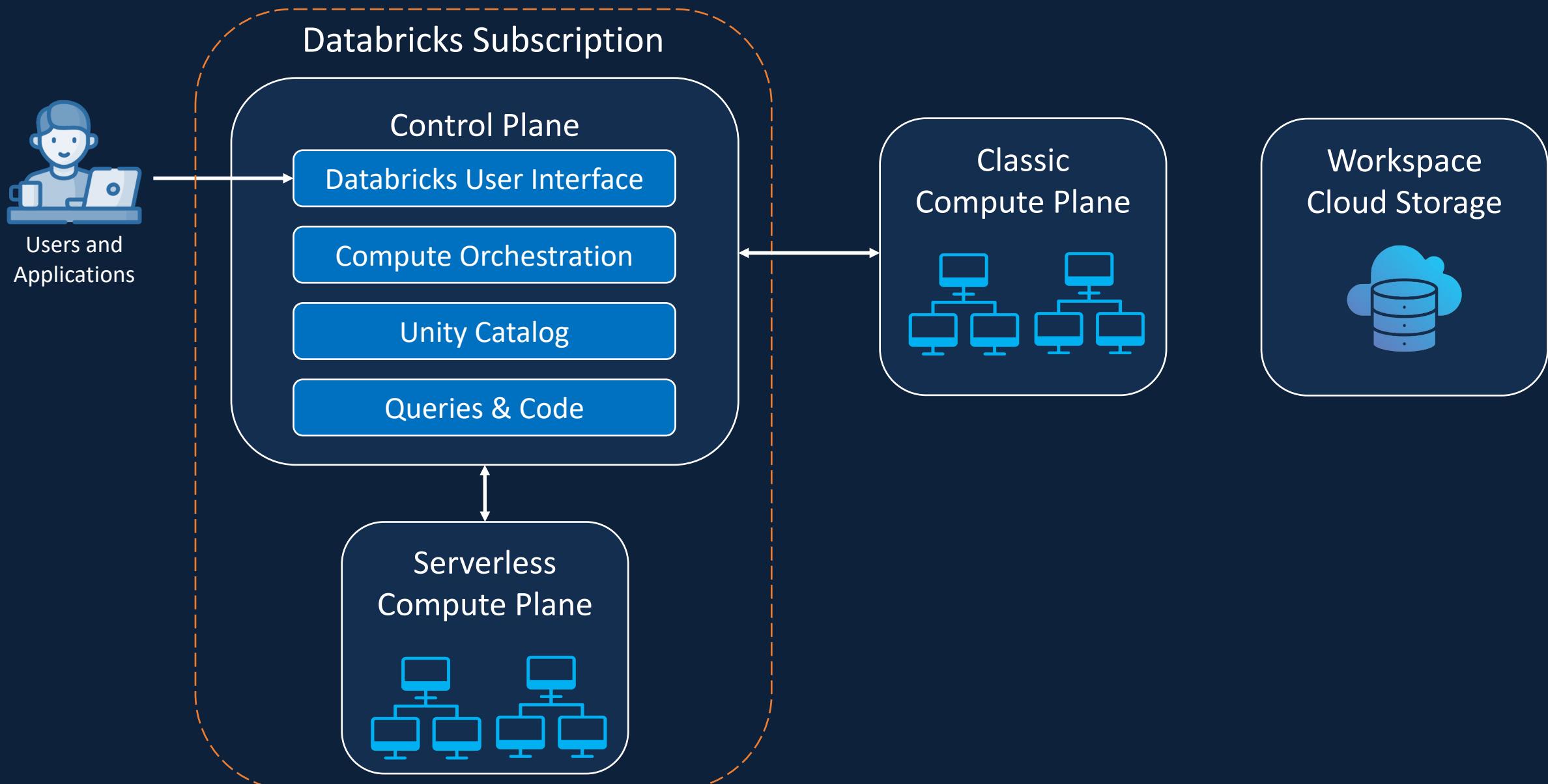


Databricks Architecture

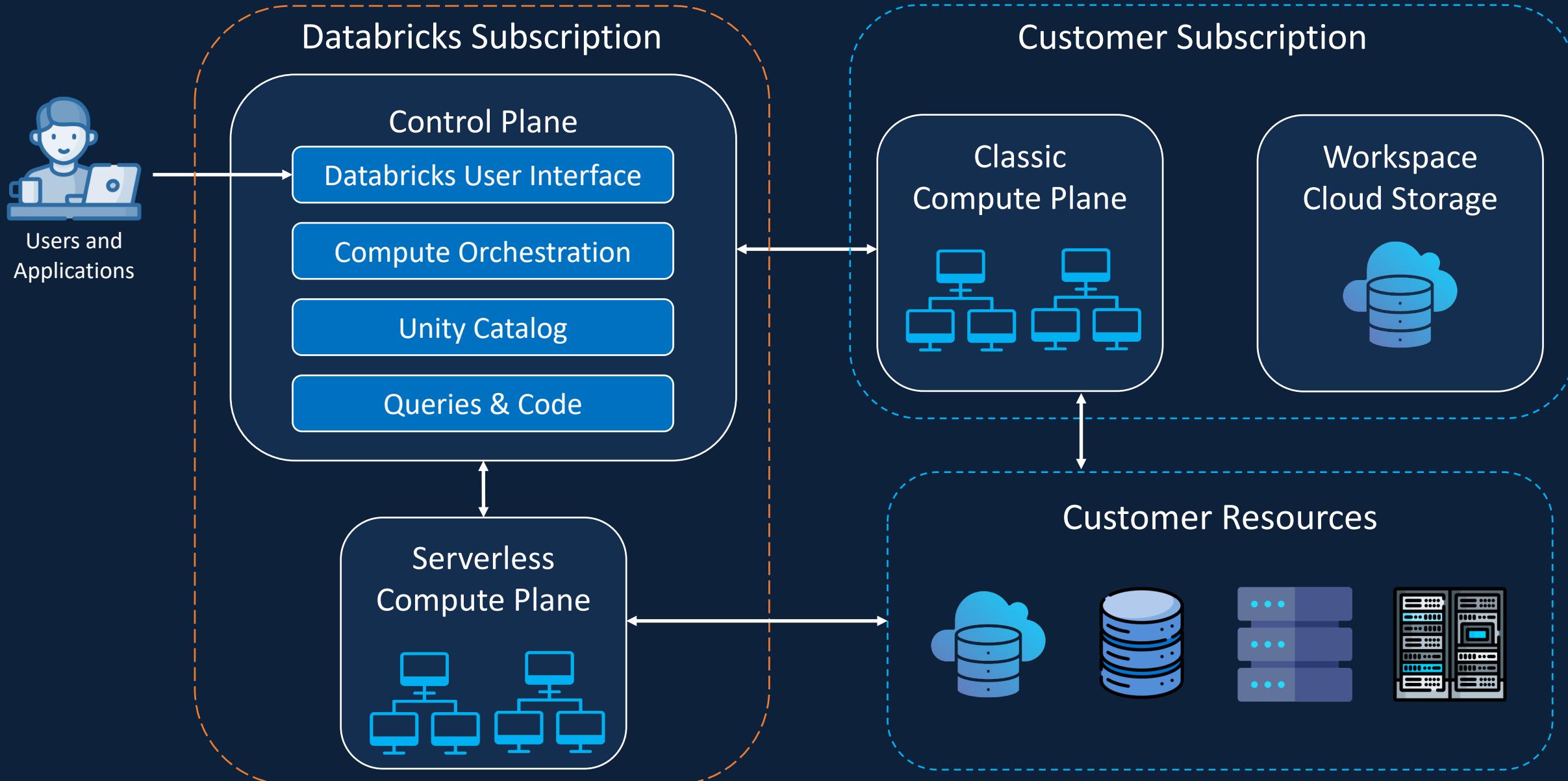
Databricks Architecture



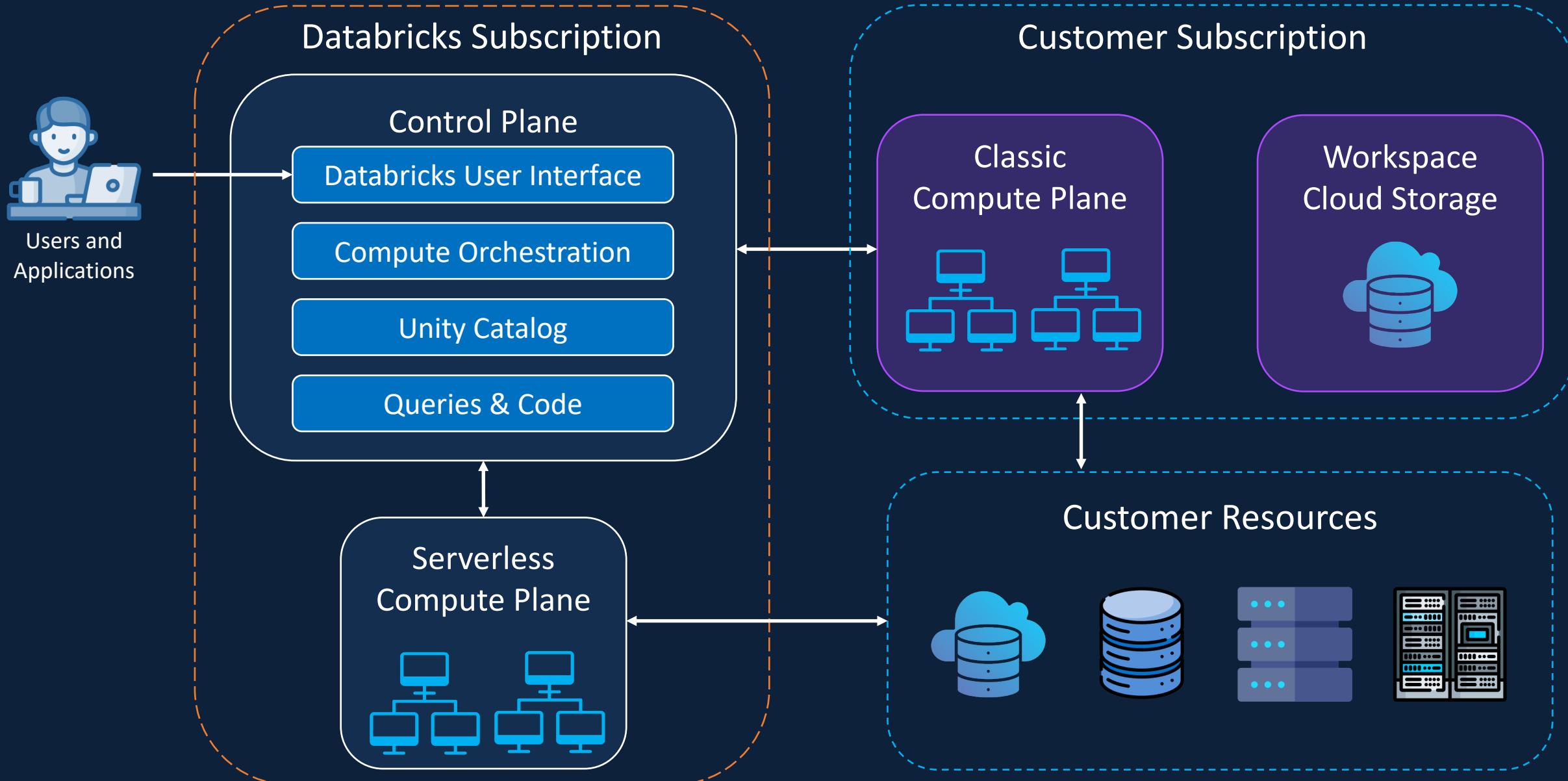
Databricks Architecture



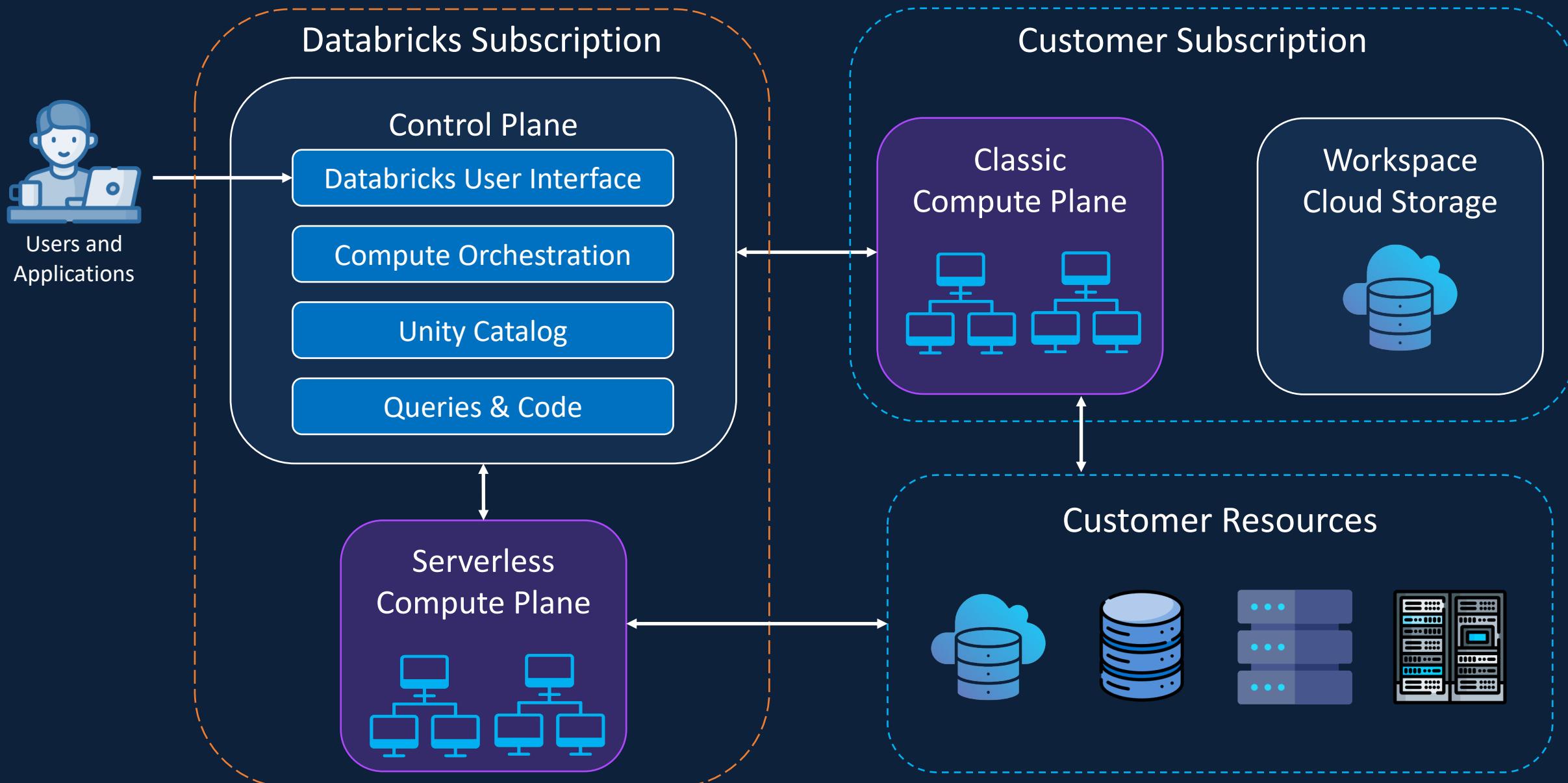
Databricks Architecture



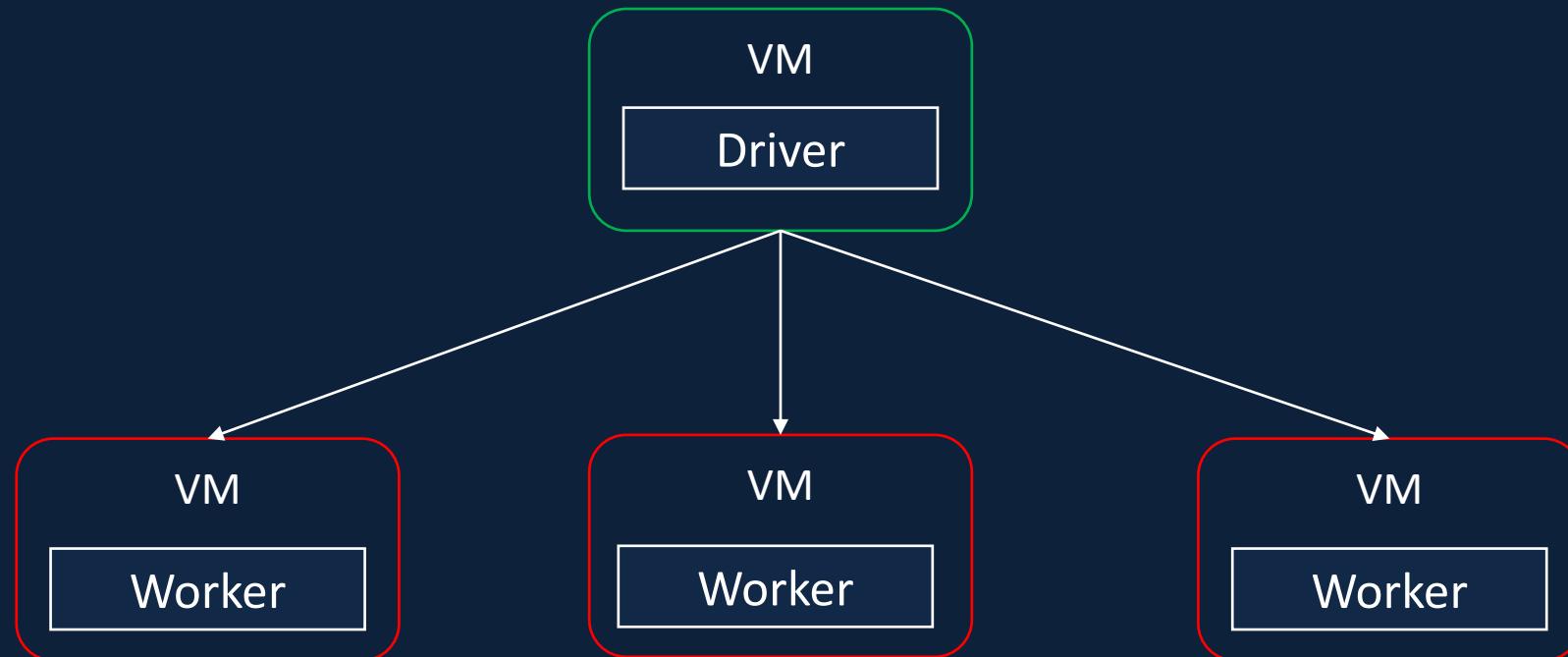
Databricks Architecture



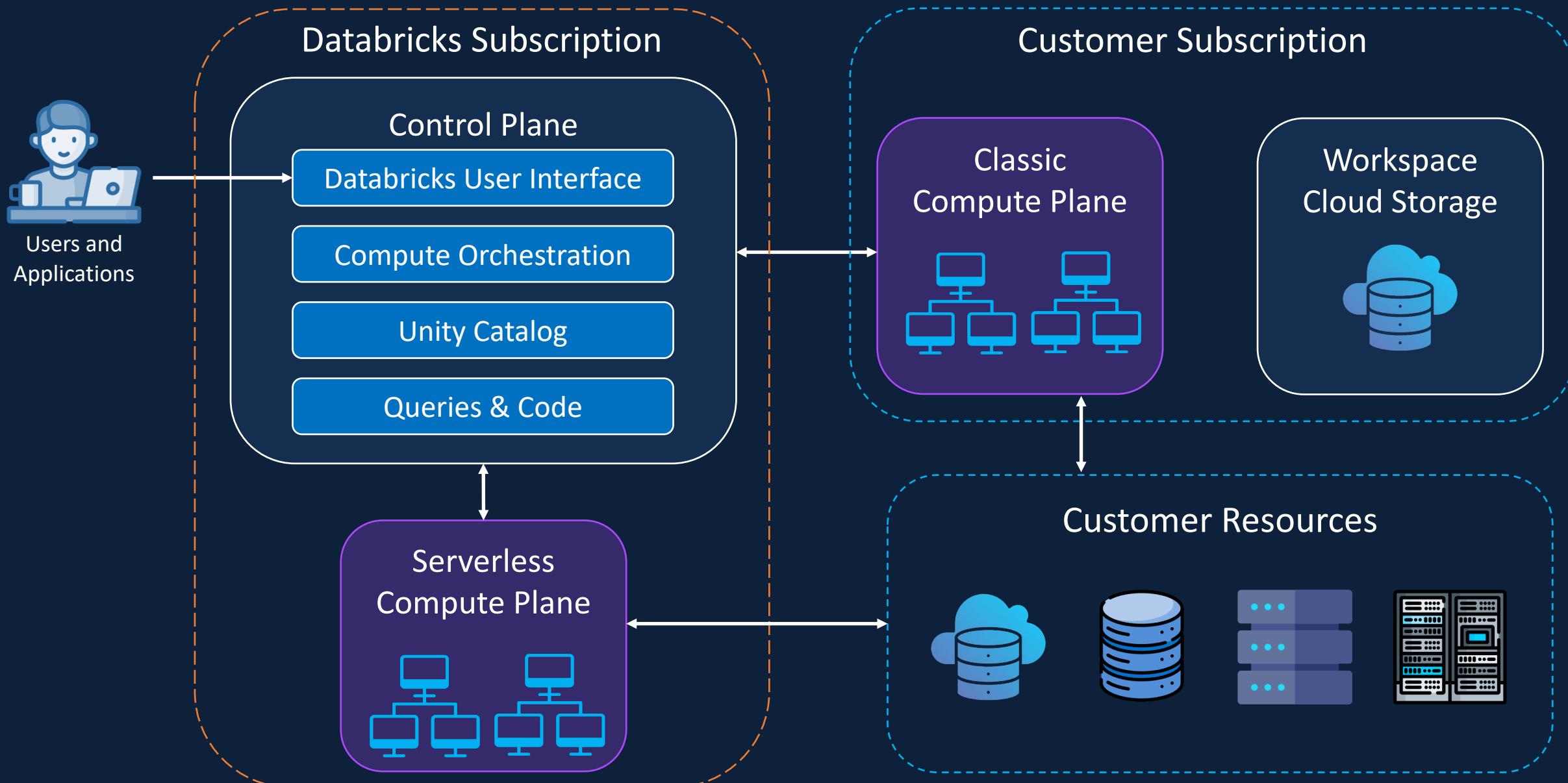
Databricks Compute



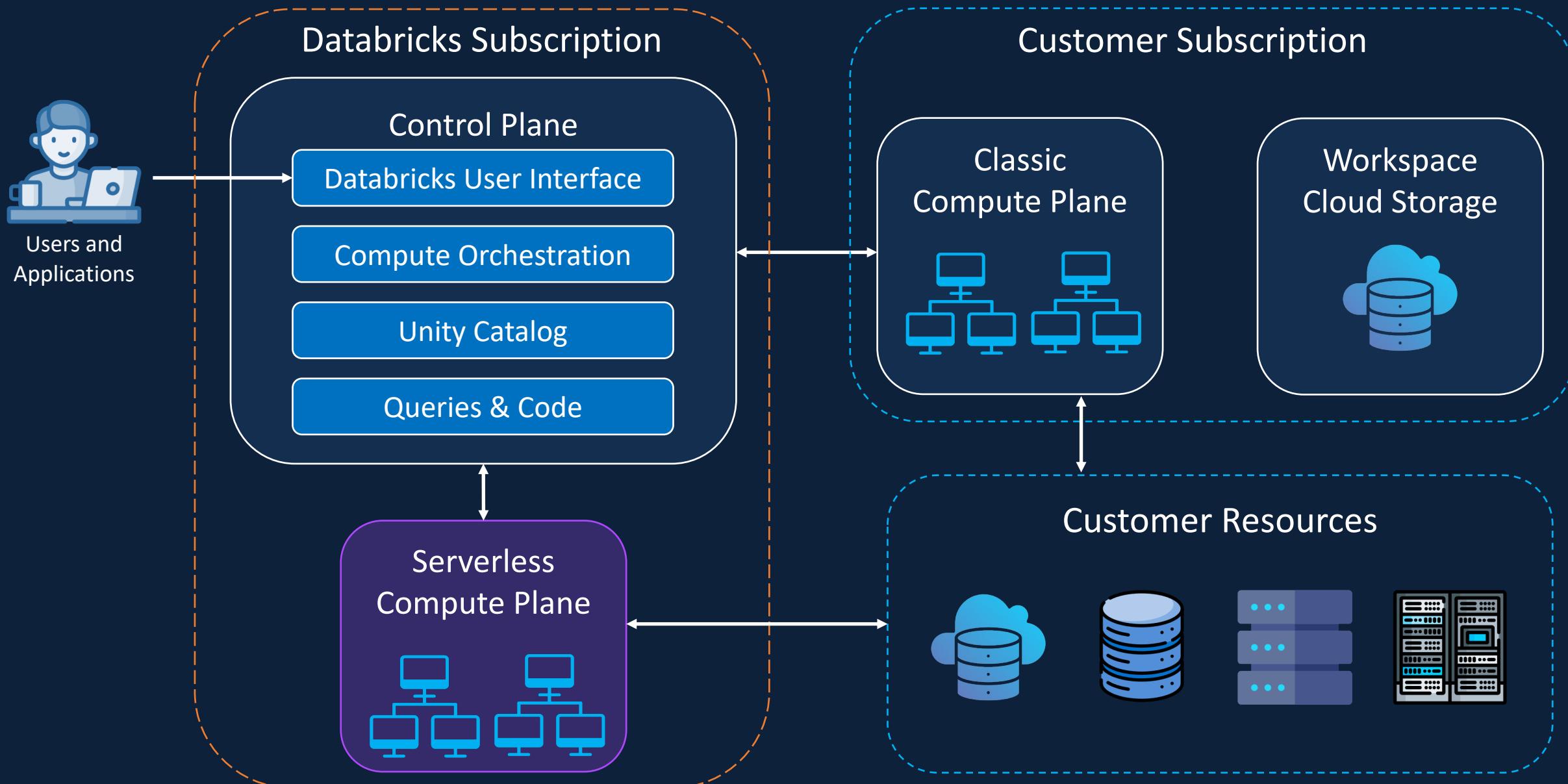
Databricks Compute



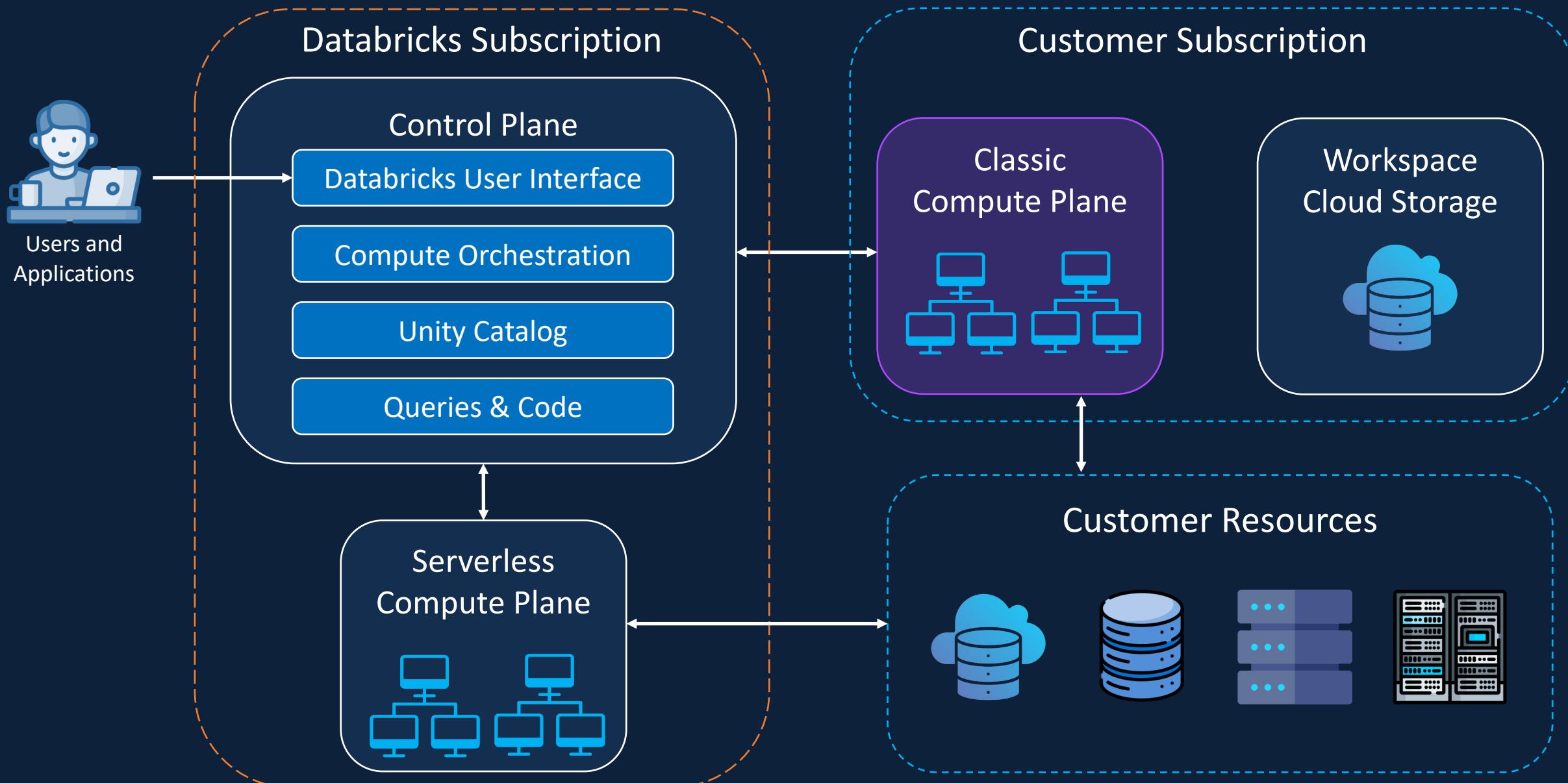
Databricks Compute



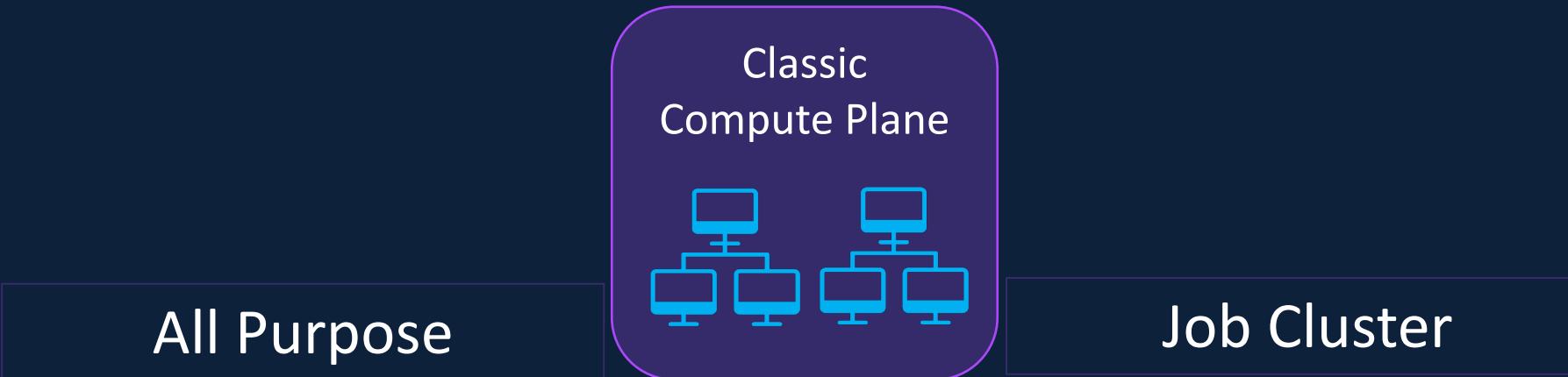
Databricks Compute



Databricks Compute



Databricks Classic Compute



Created manually

Persistent

Suitable for interactive workloads

Shared among many users

Expensive to run

Created by Jobs

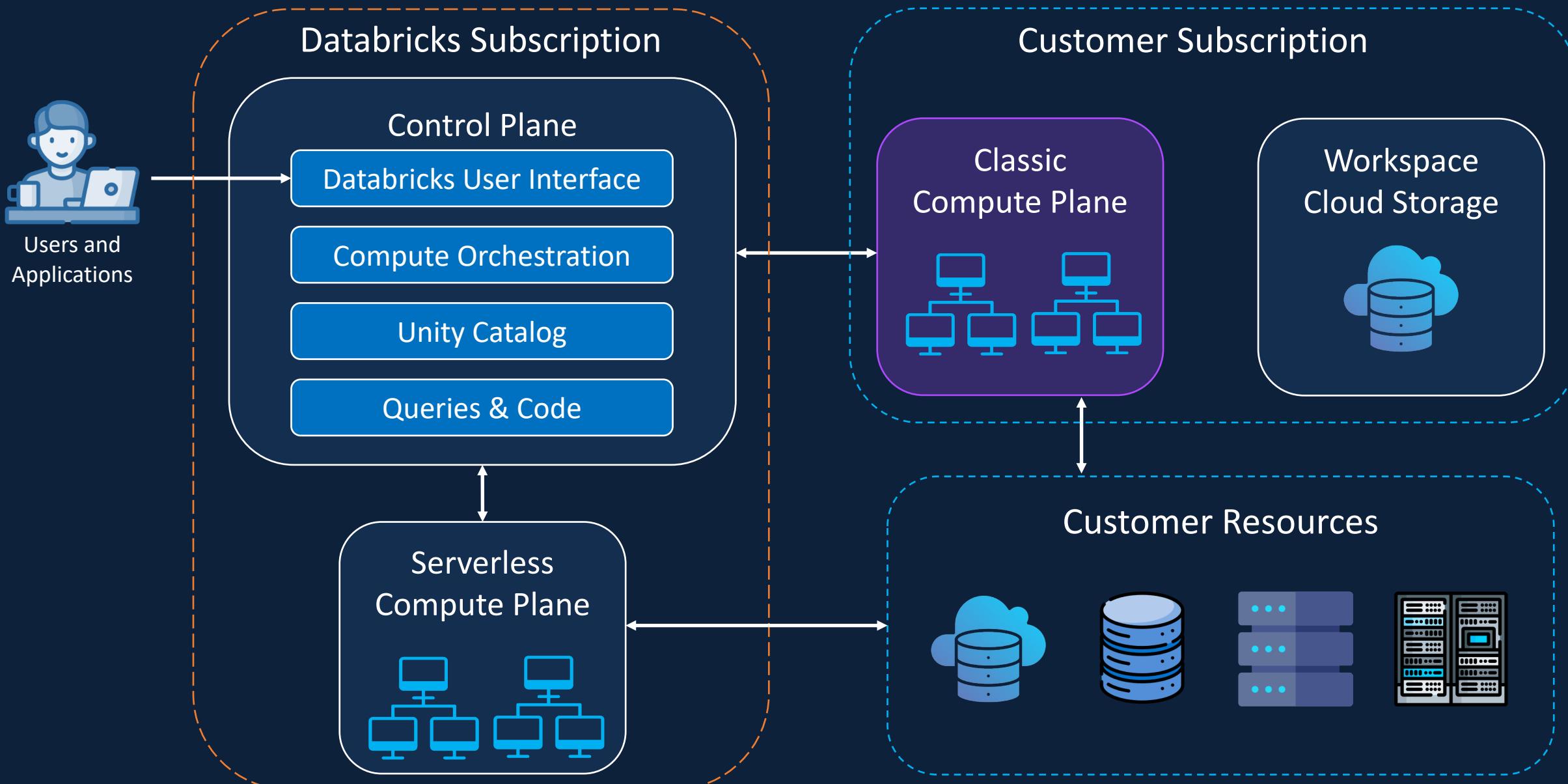
Terminated at the end of the job

Suitable for automated workloads

Isolated just for the job

Cheaper to run

Cluster Configuration



Cluster Configuration



Cluster Configuration edit

Policy ?

Unrestricted | ▼

Multi node Single node

Access mode ? **Single user access** ?

Single user | ▼ Ramesh Retnasamy (az.adm1... | ▼

Performance

Databricks runtime version ?

Runtime: 11.3 LTS (Scala 2.12, Spark 3.3.0) | ▼

Use Photon Acceleration ?

Worker type	Min workers	Max workers
Standard_DS3_v2 14 GB Memory, 4 Cores ▼	2	8

⚠ Spot instances ?

Driver type

Same as worker 14 GB Memory, 4 Cores | ▼

Enable autoscaling ?

Terminate after 120 minutes of inactivity ?

Cluster Configuration

Single/ Multi Node

Multi Node

Single Node

Cluster Configuration

Single/ Multi Node

Access Mode

Single User

Only One User Access
Supports Python, SQL, Scala, R

Shared

Multiple User Access
Only available in Premium. Supports Python, SQL

No Isolation Shared

Multiple User Access
Supports Python, SQL, Scala, R

Cluster Configuration

Single/ Multi Node

Access Mode

Databricks Runtime

Databricks Runtime

Apace
Spark

Supporting
Libraries

Photon

Databricks Runtime ML

Apace
Spark

Supporting
Libraries

Popular ML
Libraries

Photon

Cluster Configuration

Single/ Multi Node

Access Mode

Databricks Runtime

Auto Termination

Auto Termination

- Terminates the cluster after X minutes of inactivity
- Default value for Single Node and Standard clusters is 120 minutes
- Users can specify a value between 10 and 43200 mins as the duration

Cluster Configuration

Single/ Multi Node

Access Mode

Databricks Runtime

Auto Termination

Auto Scaling

Auto Scaling

- User specifies the min and max worker nodes
- Auto scales between min and max based on the workload
- Users can opt for spot instances for the worker nodes

Cluster Configuration

Single/ Multi Node

Access Mode

Databricks Runtime

Auto Termination

Auto Scaling

Cluster VM Type/ Size

Memory Optimized

Compute Optimized

Storage Optimized

General Purpose

GPU Accelerated

Cluster Configuration

Single/ Multi Node

Access Mode

Databricks Runtime

Auto Termination

Auto Scaling

Cluster VM Type/ Size

Cluster Policy

Cluster Configuration 

Policy 

Unrestricted 

Multi node Single node

Access mode  **Single user access** 

Single user  Ramesh Retnasamy (az.adm1... 

Performance

Databricks runtime version 

Runtime: 11.3 LTS (Scala 2.12, Spark 3.3.0) 

Use Photon Acceleration 

Worker type  **Min workers**  **Max workers** 

Standard_DS3_v2  14 GB Memory, 4 Cores  2  8  Spot instances 

Driver type

Same as worker  14 GB Memory, 4 Cores 

Enable autoscaling 

Terminate after  120 minutes of inactivity 

Cluster Configuration

Single/ Multi Node

Access Mode

Databricks Runtime

Auto Termination

Auto Scaling

Cluster VM Type/ Size

Cluster Policy

Cluster Configuration 

Policy 

Personal Compute 

Single user access 

Ramesh Retnasamy (az.adm1@outlook.com) 

Performance

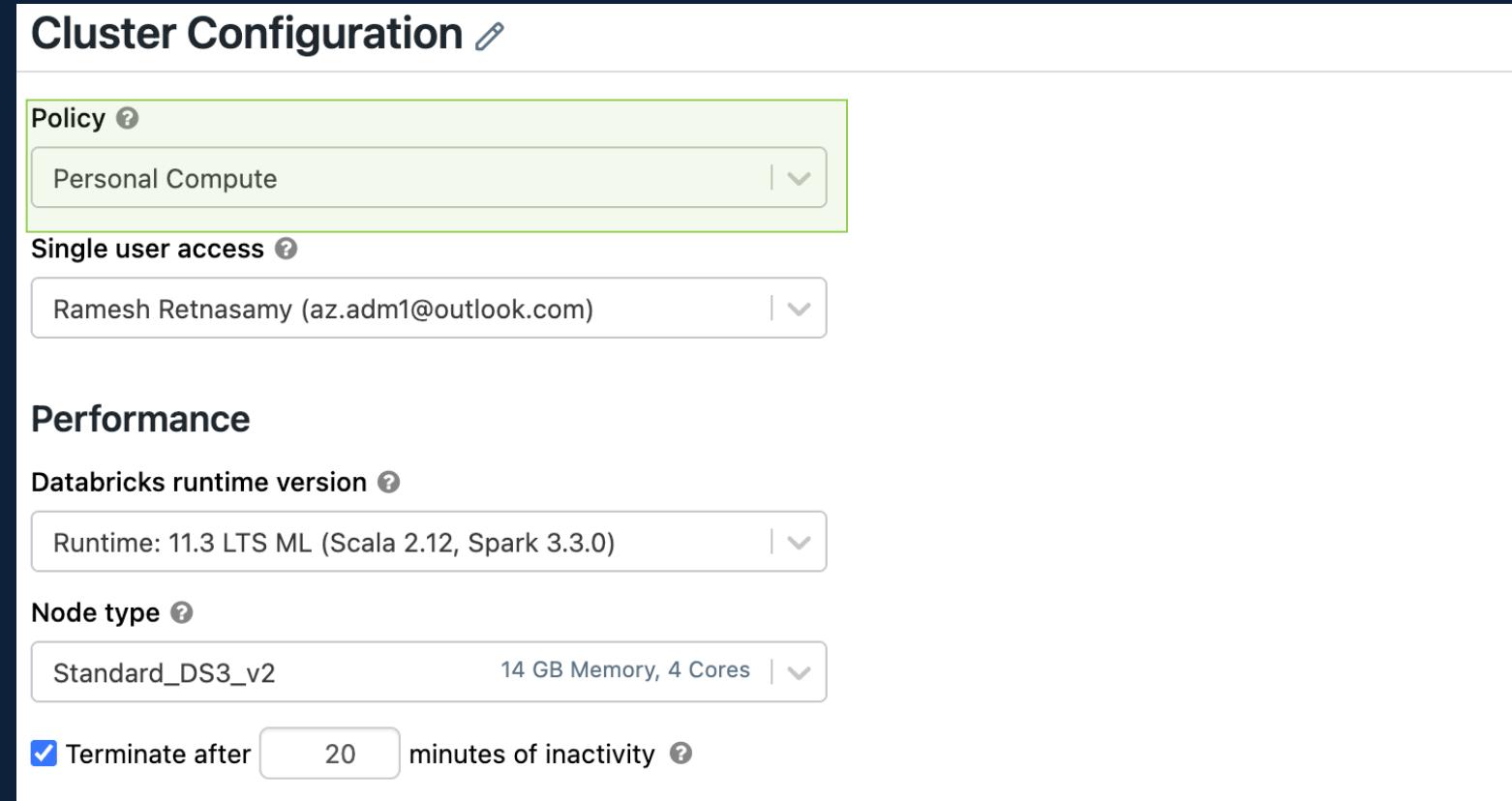
Databricks runtime version 

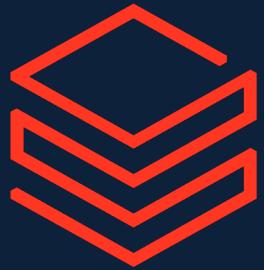
Runtime: 11.3 LTS ML (Scala 2.12, Spark 3.3.0) 

Node type 

Standard_DS3_v2 14 GB Memory, 4 Cores 

Terminate after minutes of inactivity 





Create Databricks Compute



Databricks Notebooks



Databricks Magic Commands

Databricks Magic Commands

%python, %sql, %scala, %r

Switch to a different language for a specific cell

%md

Markdown for documenting notebooks

%fs

Run file system commands

%sh

Run shell commands (Driver Node only)

%pip

Install Python libraries

%run

Include/ Import another notebook into the current notebook



Databricks Utilities

Databricks Utilities

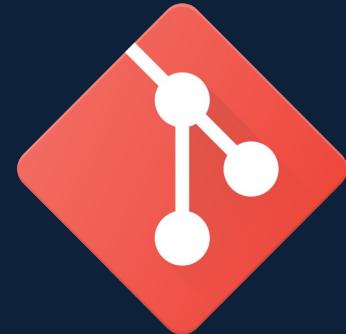


File System Utilities

Secrets Utilities

Widget Utilities

Notebook Workflow Utilities



Databricks Git Folders / Repos

Git Folders is a visual Git client made available in the Databricks workspace to enable collaborative development amongst developers.

Notebook Version History



1. Notebooks Introduction Python

Last edit was 5 days ago

You are viewing a version from Nov 8 2024, 14:59 PM GMT. [Exit](#)

1

```
%md  
# Introduction to Notebooks  
## Heading 2  
- bullet 1  
- bullet 2
```

This notebook gives you an introduction to databricks notebooks!

2

```
1print('hello')  
2print('bye')
```

hello
bye

3

```
1print('Hello Again')
```

Hello Again

Run all Terminated Schedule Share

Version history

Date	Author	Actions
Nov 8 2024, 14:59 PM GMT	Ramesh Retnasamy	Save now
Nov 8 2024, 14:55 PM GMT	Ramesh Retnasamy	
Nov 8 2024, 14:53 PM GMT	Ramesh Retnasamy	
Nov 8 2024, 14:52 PM GMT	Ramesh Retnasamy	
Nov 8 2024, 14:50 PM GMT	Ramesh Retnasamy	
Nov 8 2024, 14:48 PM GMT	Ramesh Retnasamy	
Nov 8 2024, 14:44 PM GMT	Ramesh Retnasamy	
Nov 8 2024, 14:39 PM GMT	Ramesh Retnasamy	
Nov 8 2024, 14:36 PM GMT	Ramesh Retnasamy	

Notebook Version History

Limitations



Limited to Single Notebook

Basic Tracking Only / No Branching

Lack of integration with CI/CD pipelines

Databricks Git Folders / Repos



Holistic Version Control

Team Collaboration

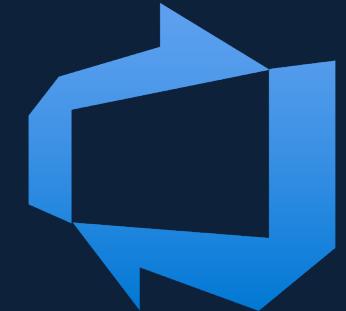
Automated CI/CD Pipelines

Databricks Git Folders / Repos



Databricks GitHub App

Personal Access Token



Databricks Git Folders / Repos



Databricks GitHub App



Link GitHub Account

Create Git Folders & Clone Git Repo

Work with Development Branches

Commit and Push Changes

Install & Configure Databricks App

Create Pull Request

Review & Merge Changes

Databricks Git Folders / Repos - Demo



Databricks GitHub App



- 1 Link GitHub Account
- 2 Install & Configure Databricks App
- 3 Create Git Folders & Clone Git Repo
- 4 Work with Development Branches
- 5 Commit and Push Changes
- 6 Create Pull Request
- 7 Review & Merge Changes

Unity Catalog - Introduction



Unity Catalog Overview

Enable Unity Catalog Metastore

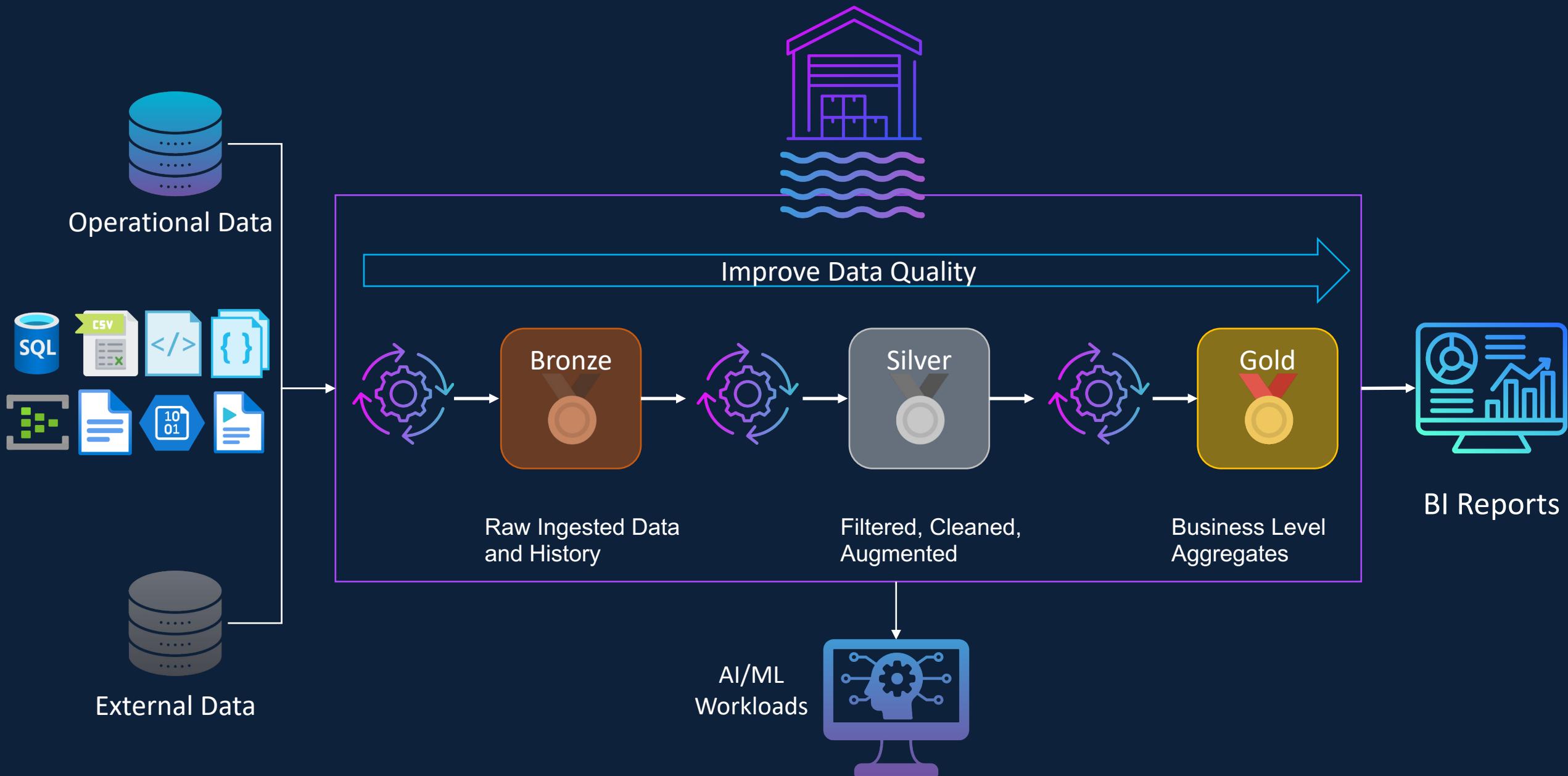
Cluster Configurations

Unity Catalog Object Model

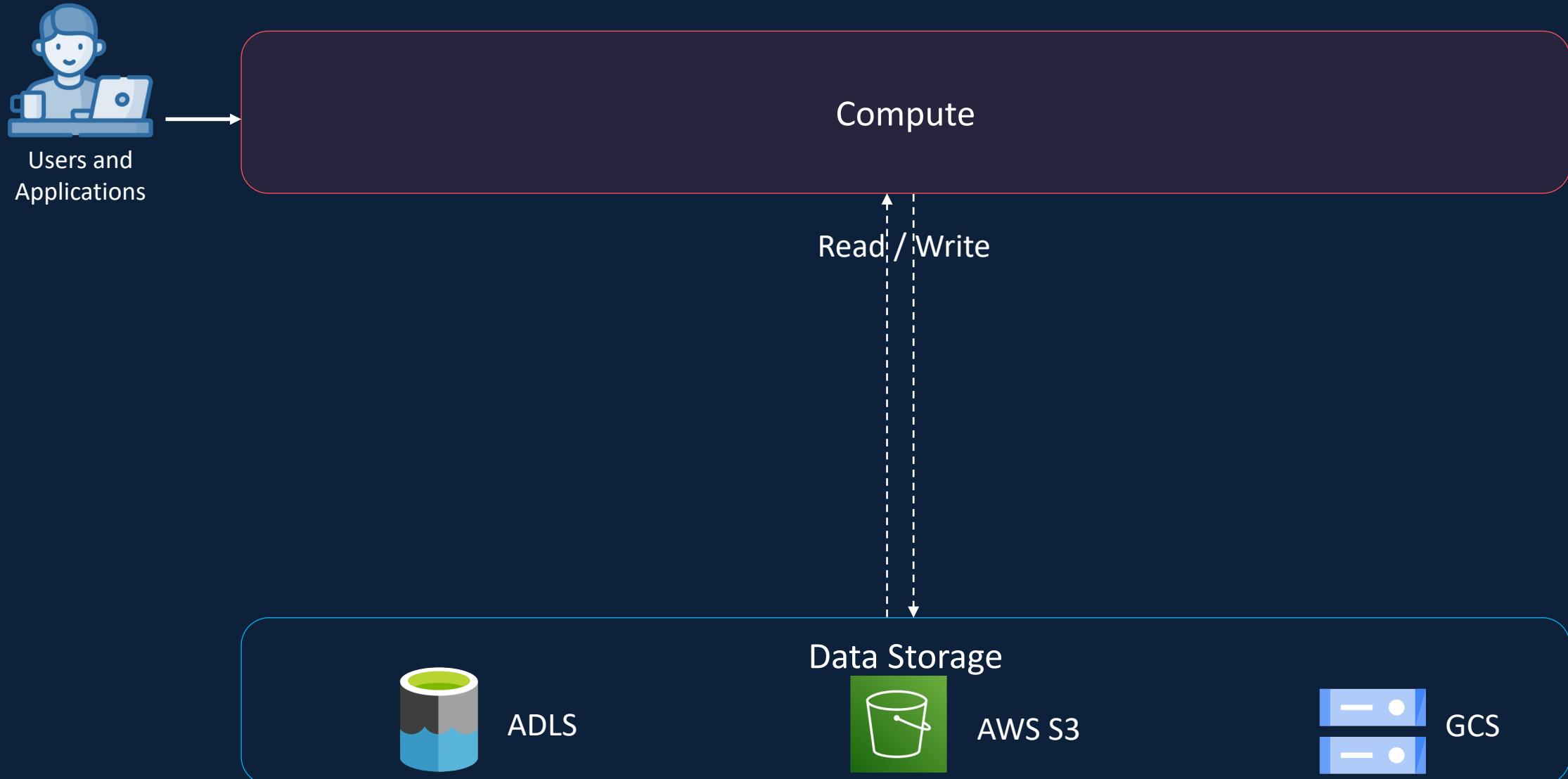
Access External Data Lake

Introduction to Unity Catalog

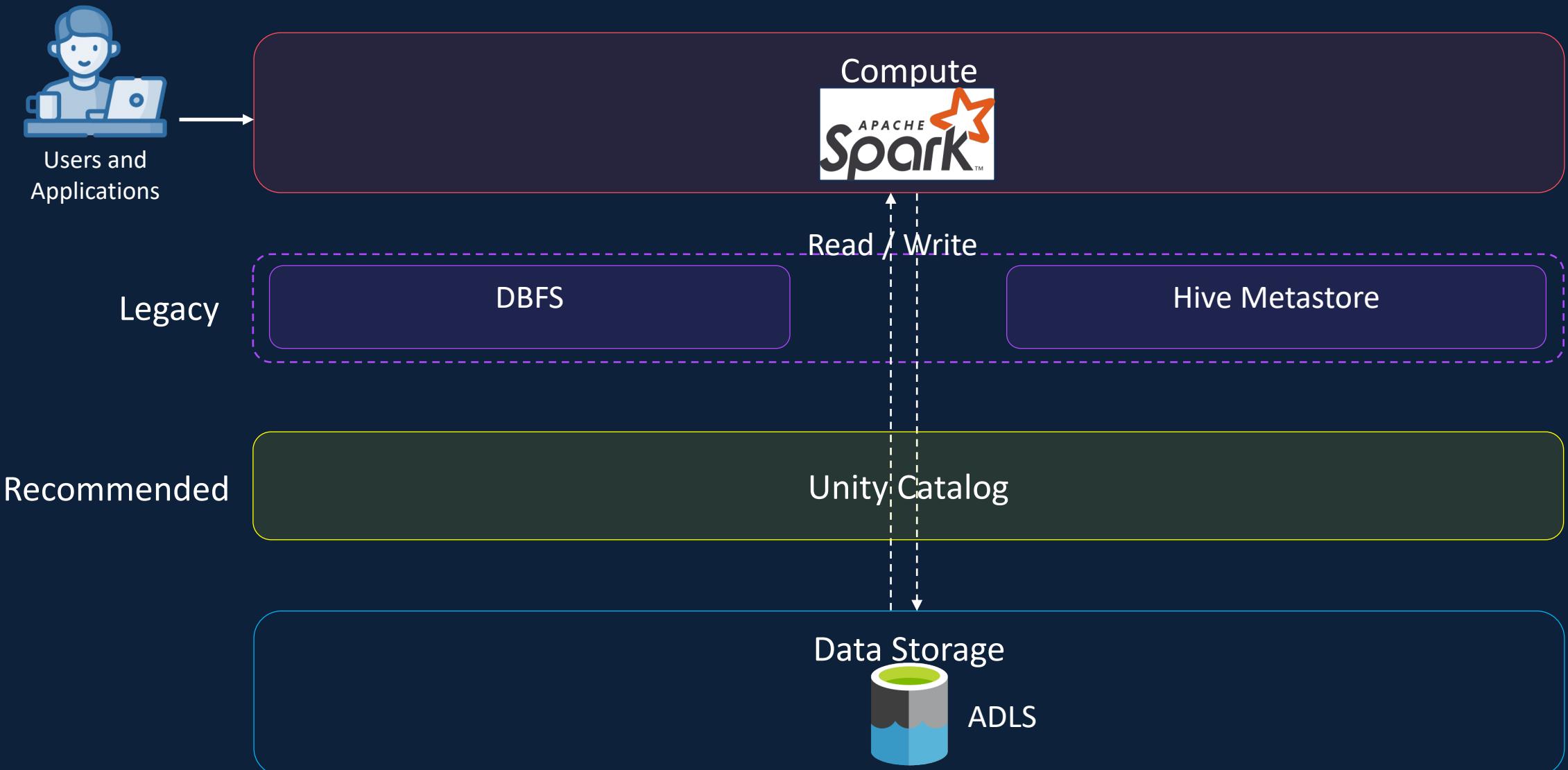
Medallion Architecture



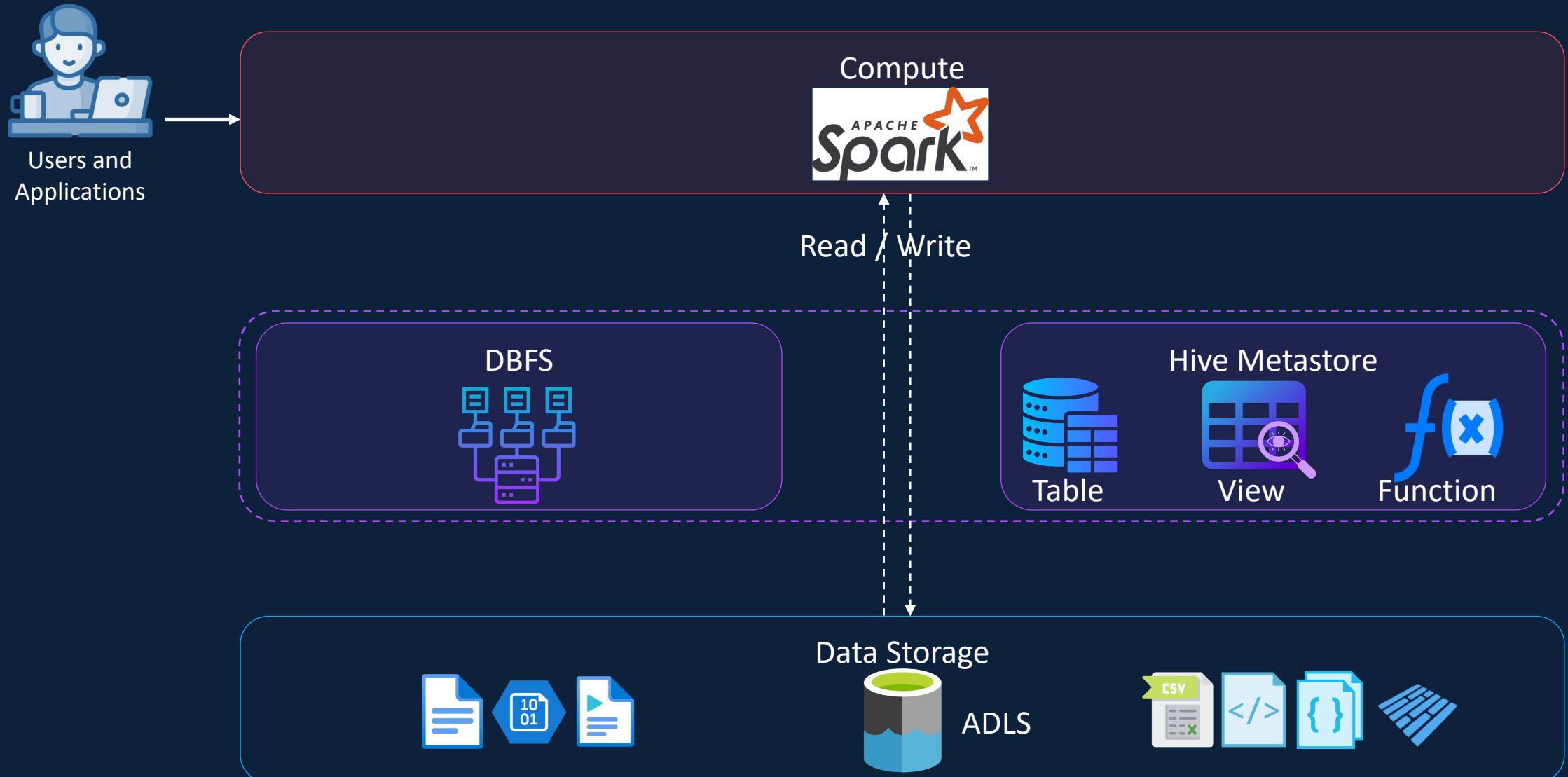
Data Access Patterns



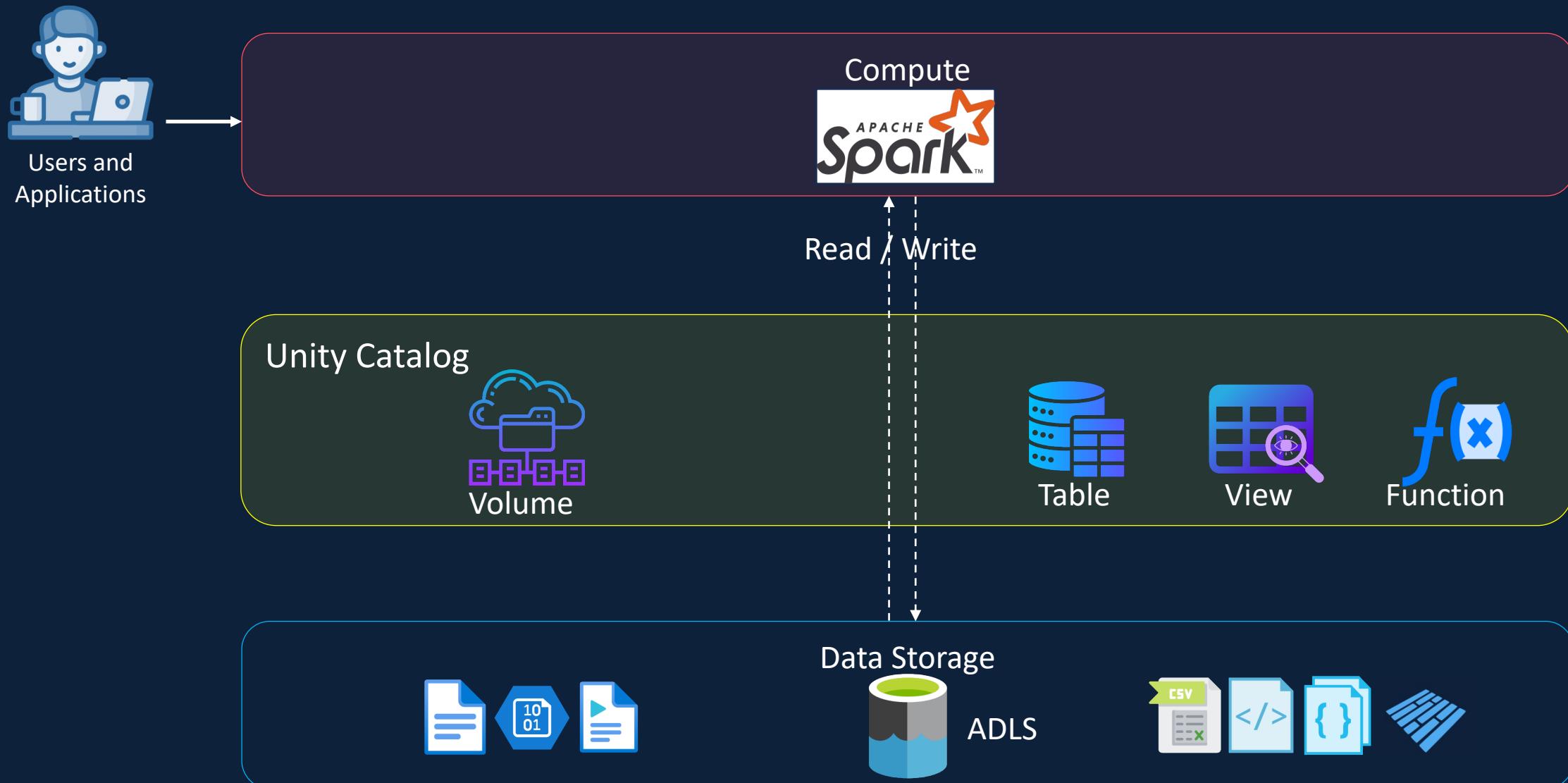
Data Access Patterns



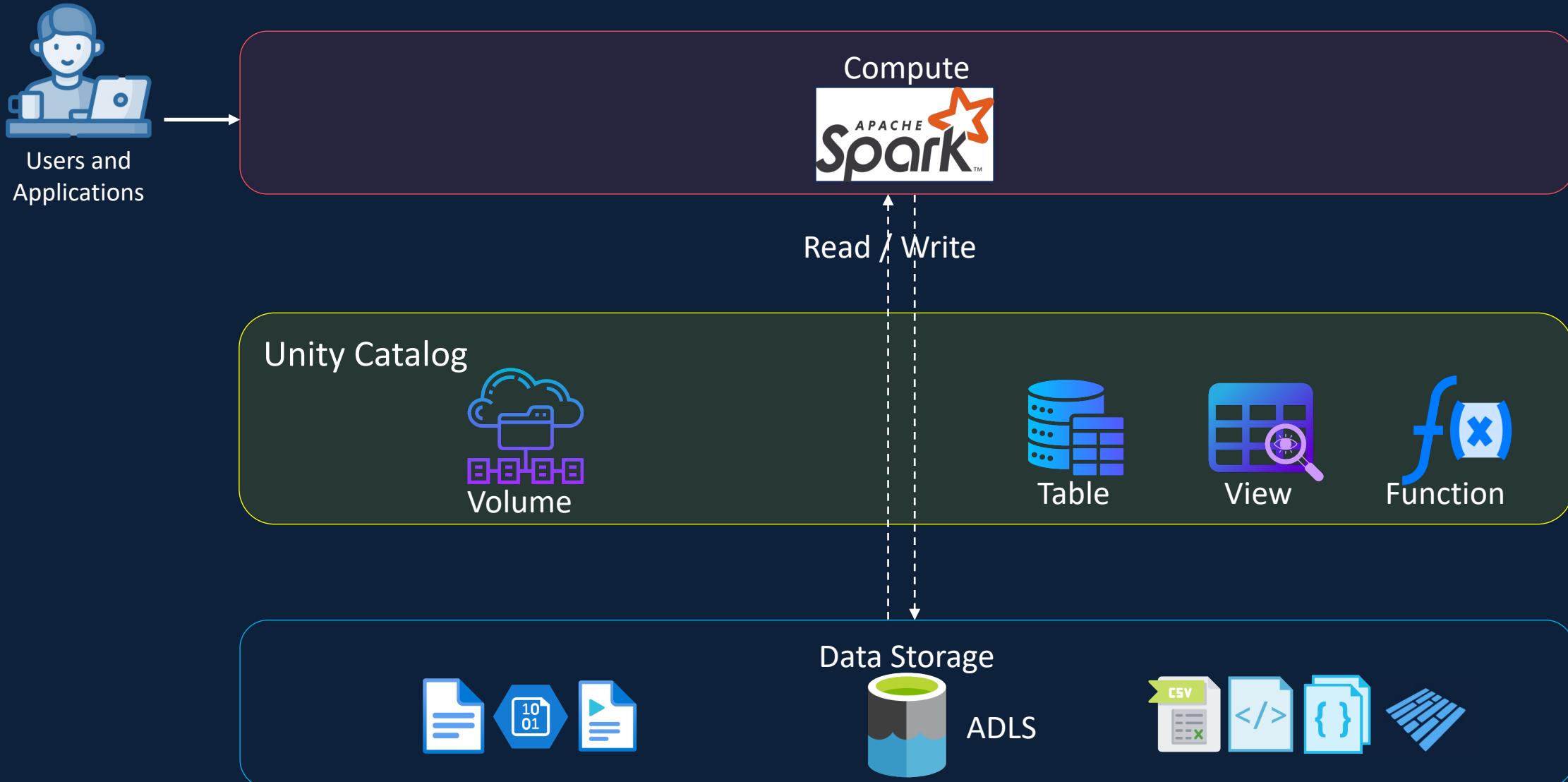
Data Access Patterns



Data Access Patterns



Unity Catalog Object Model



Unity Catalog Object Model

Metastore

Top level container

Only one per region

Paired with Default ADLS Storage

Unity Catalog Object Model



Unity Catalog Object Model

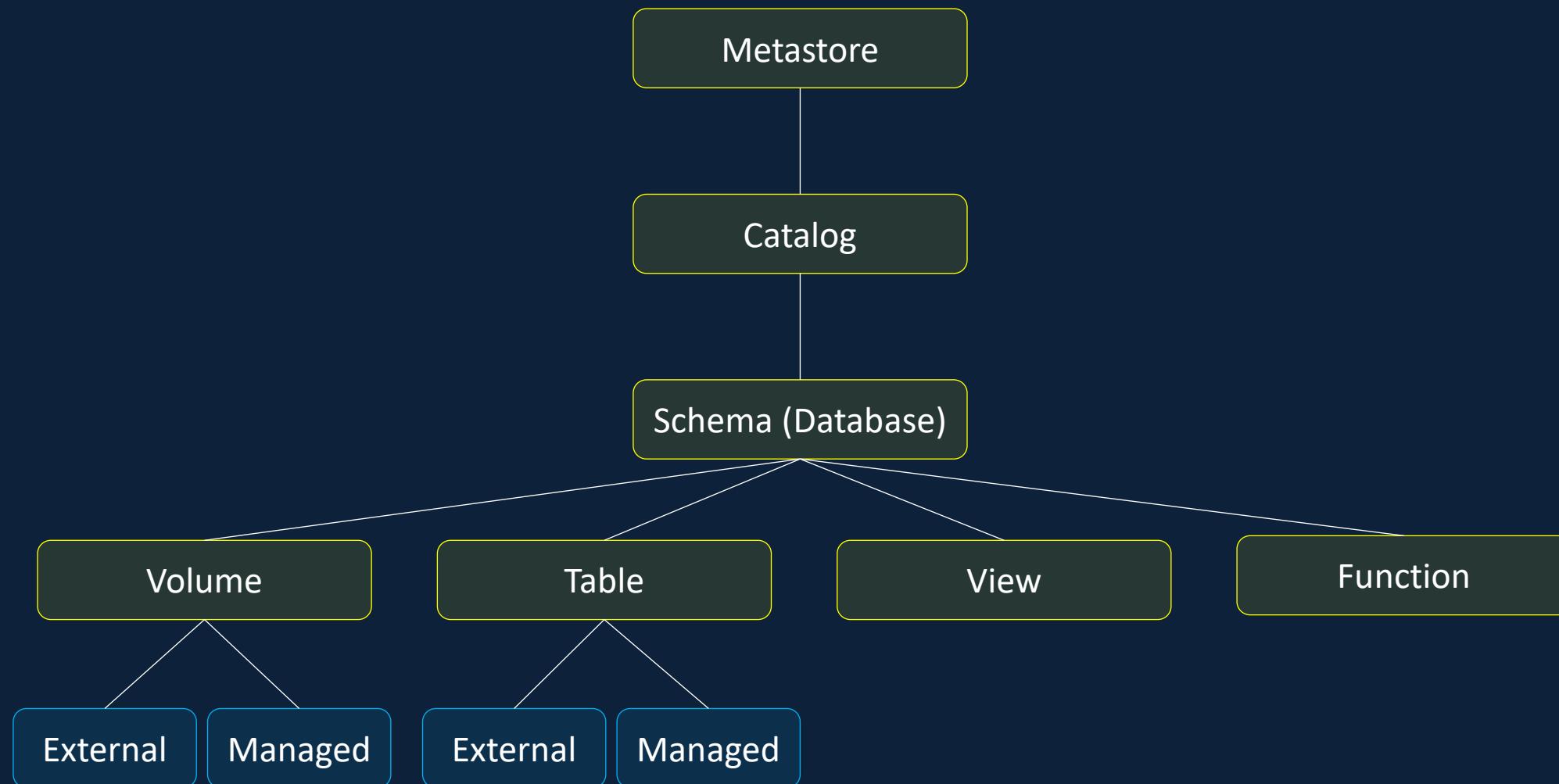


Next level container within Catalogs

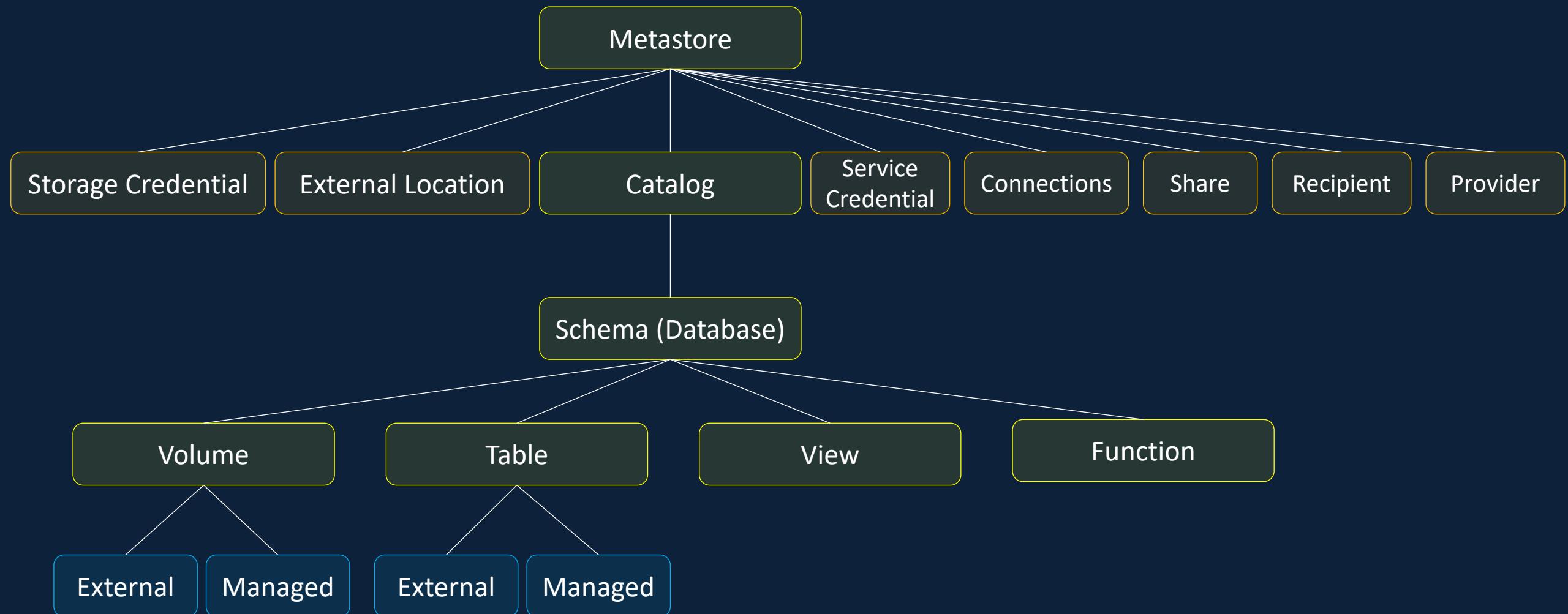
Schemas and Databases are the same

Use Schema instead of Database

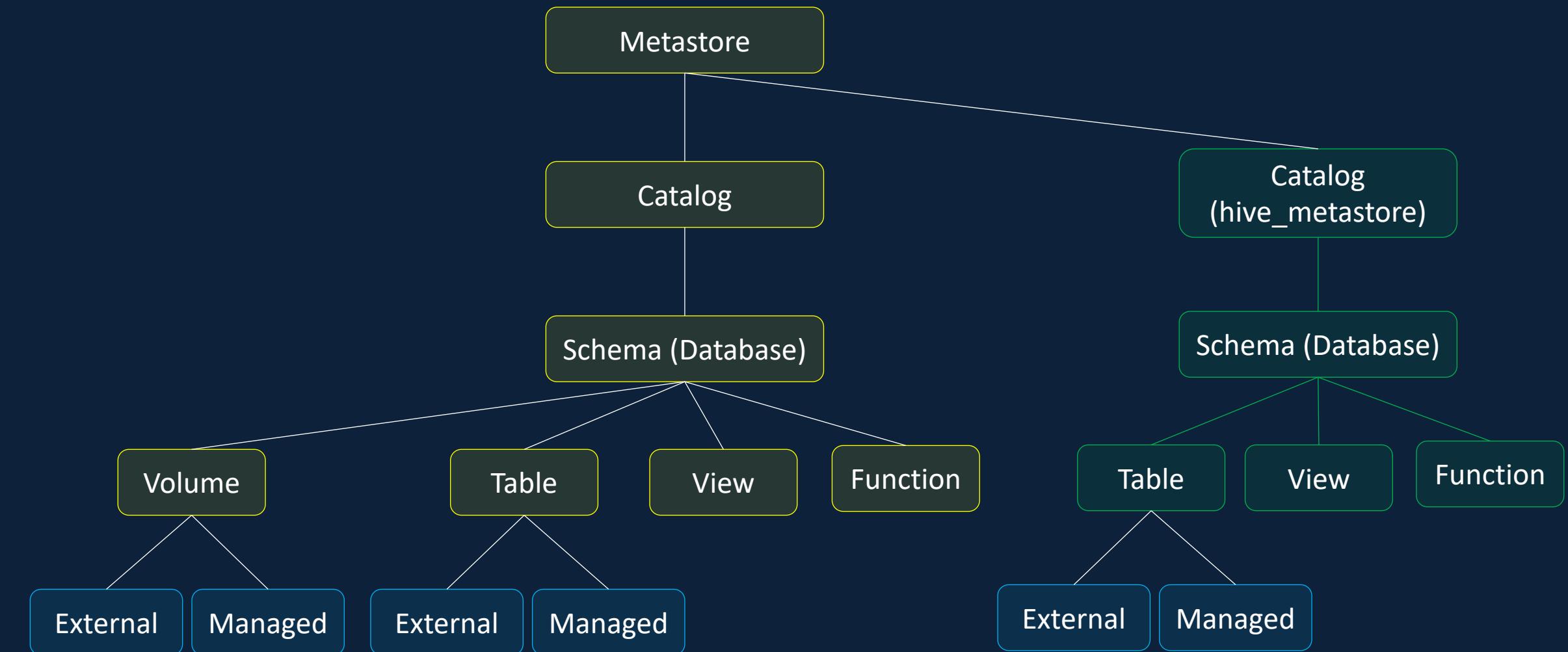
Unity Catalog Object Model



Unity Catalog Object Model

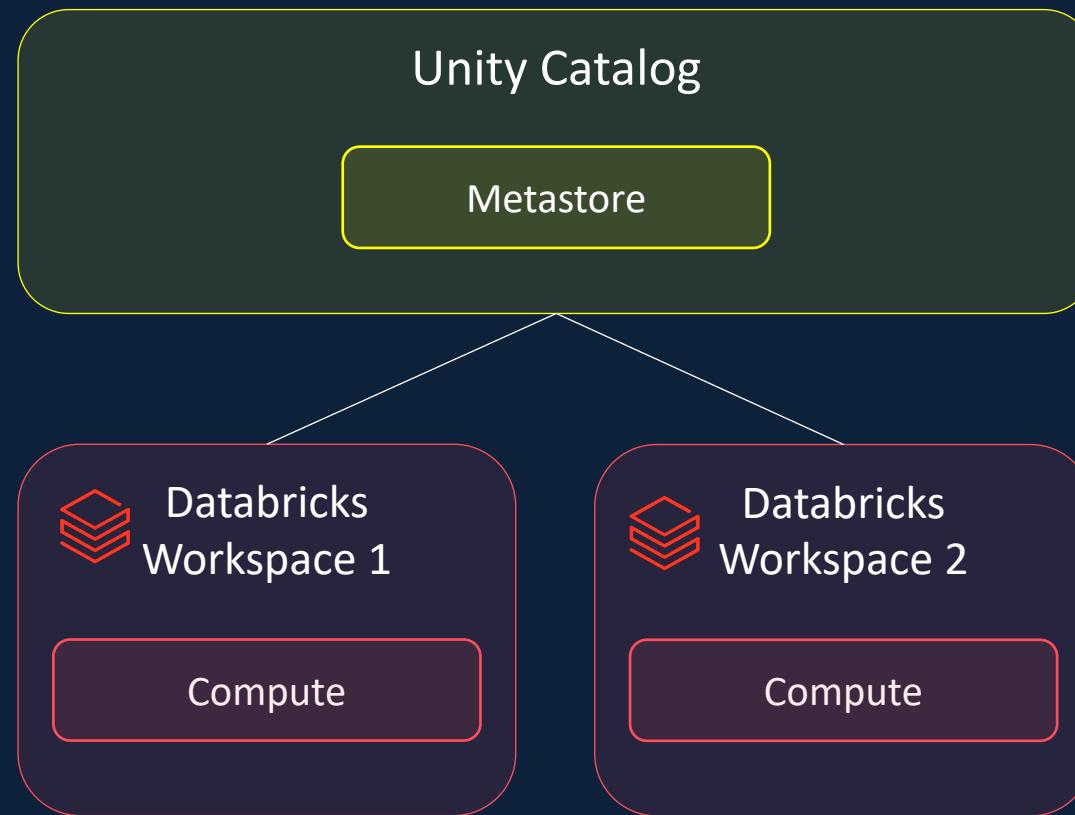


Unity Catalog / Hive Metastore Object Model

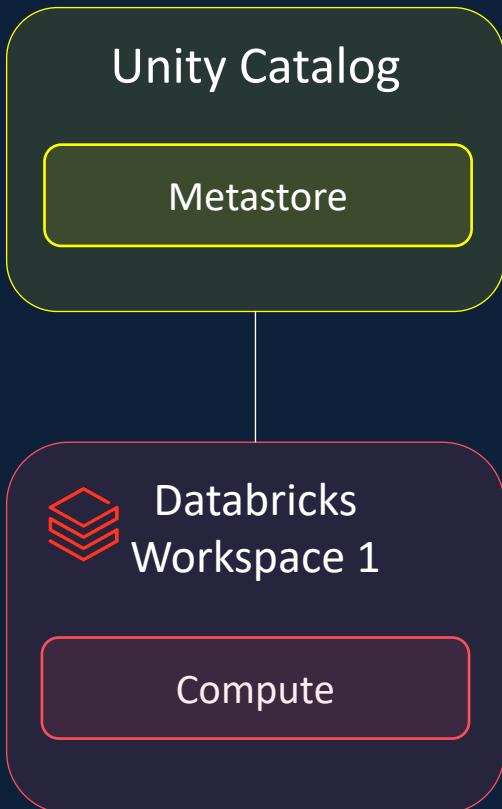


Unity Catalog Set-up

Unity Catalog Set-up



Unity Catalog Set-up



Login to Databricks Account Console

<https://accounts.azuredatabricks.net/>

User Must have Global Administrator Privileges

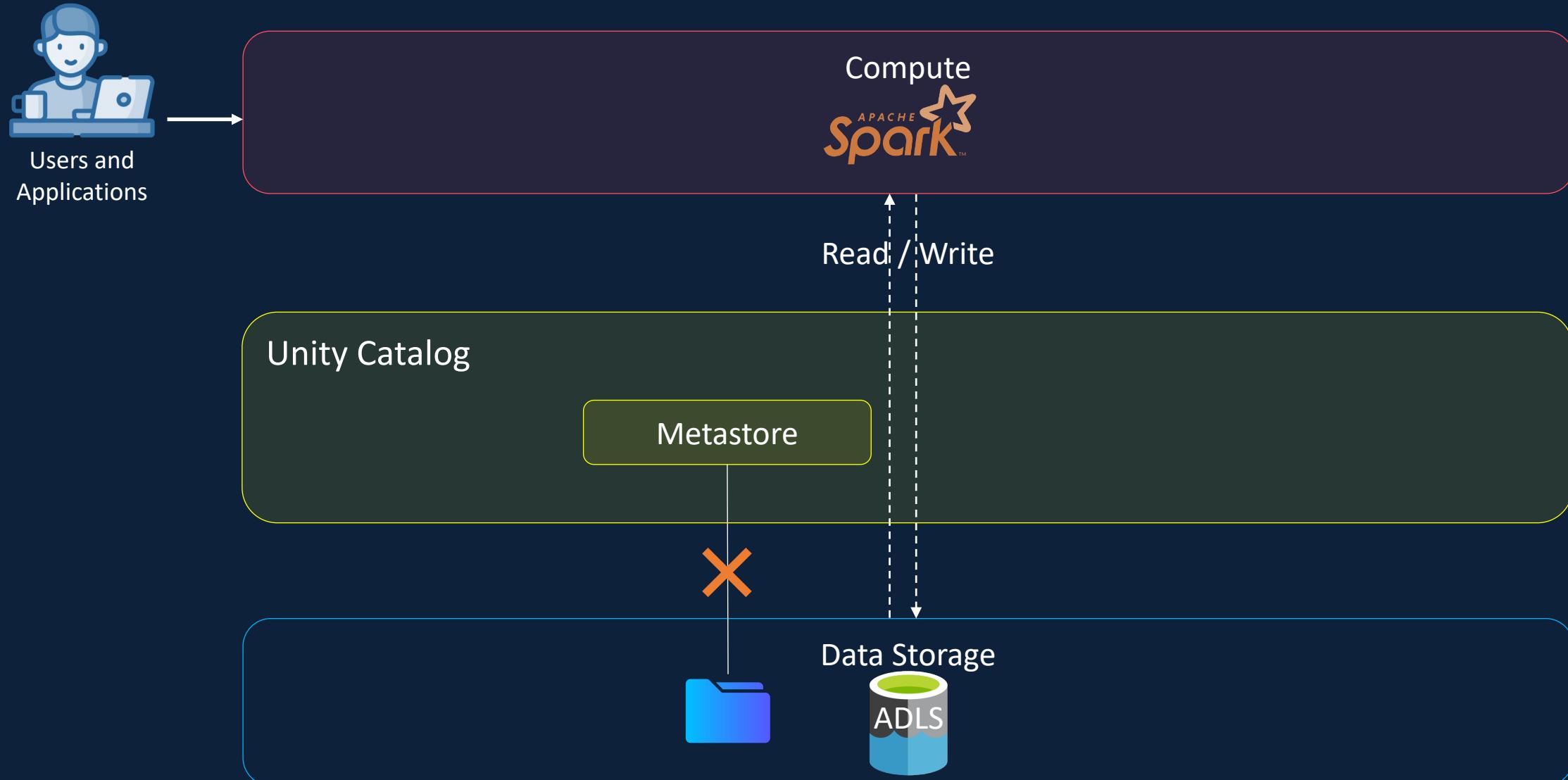
Create Unity Catalog Metastore

Assign Databricks Workspace to the Unity Catalog Metastore

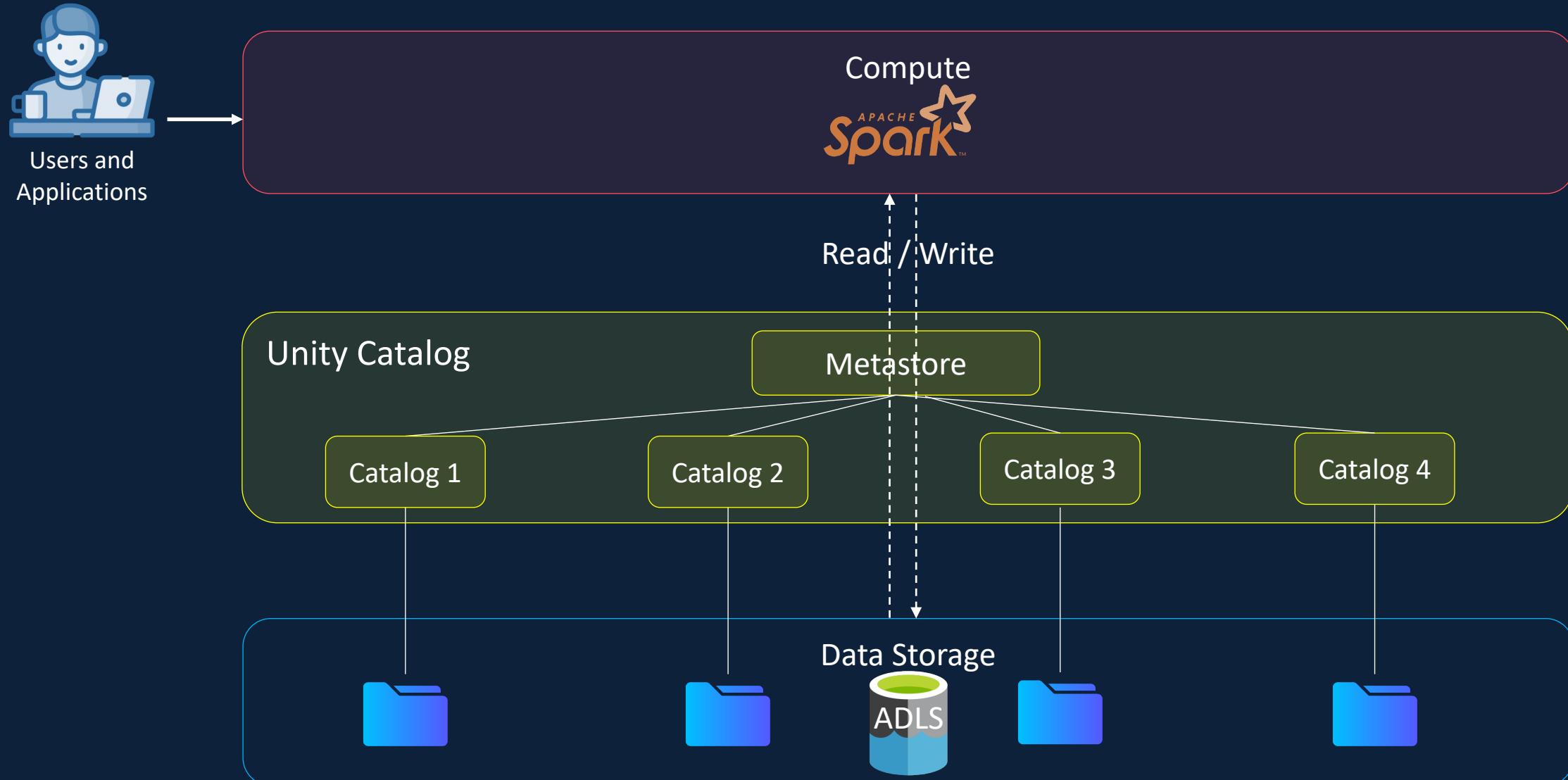
Azure Subscriptions Created on or after Nov 2023 have Unity Catalog Metastore Created by Default.

Cluster Configuration

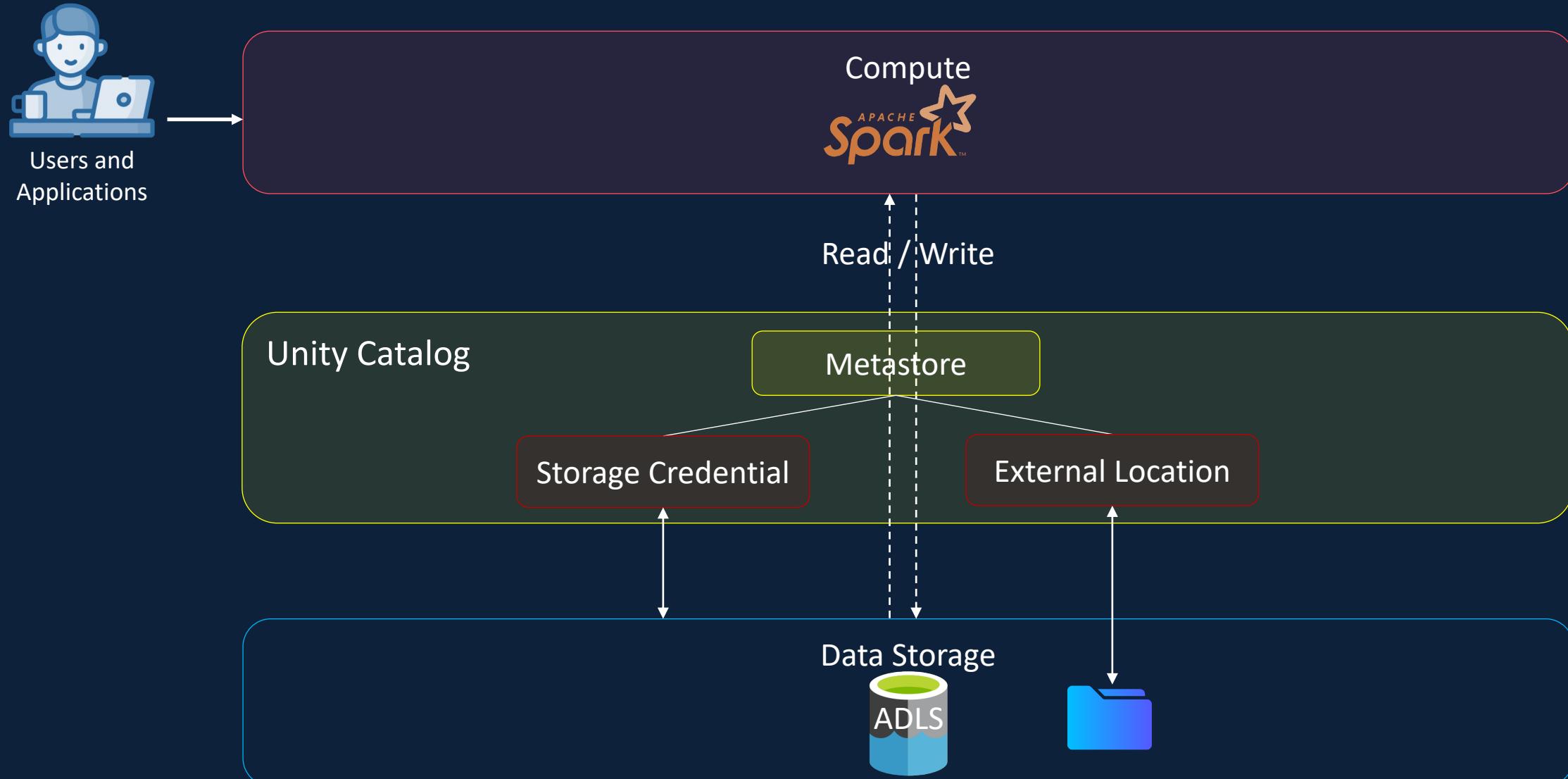
Configure Access to Cloud Storage



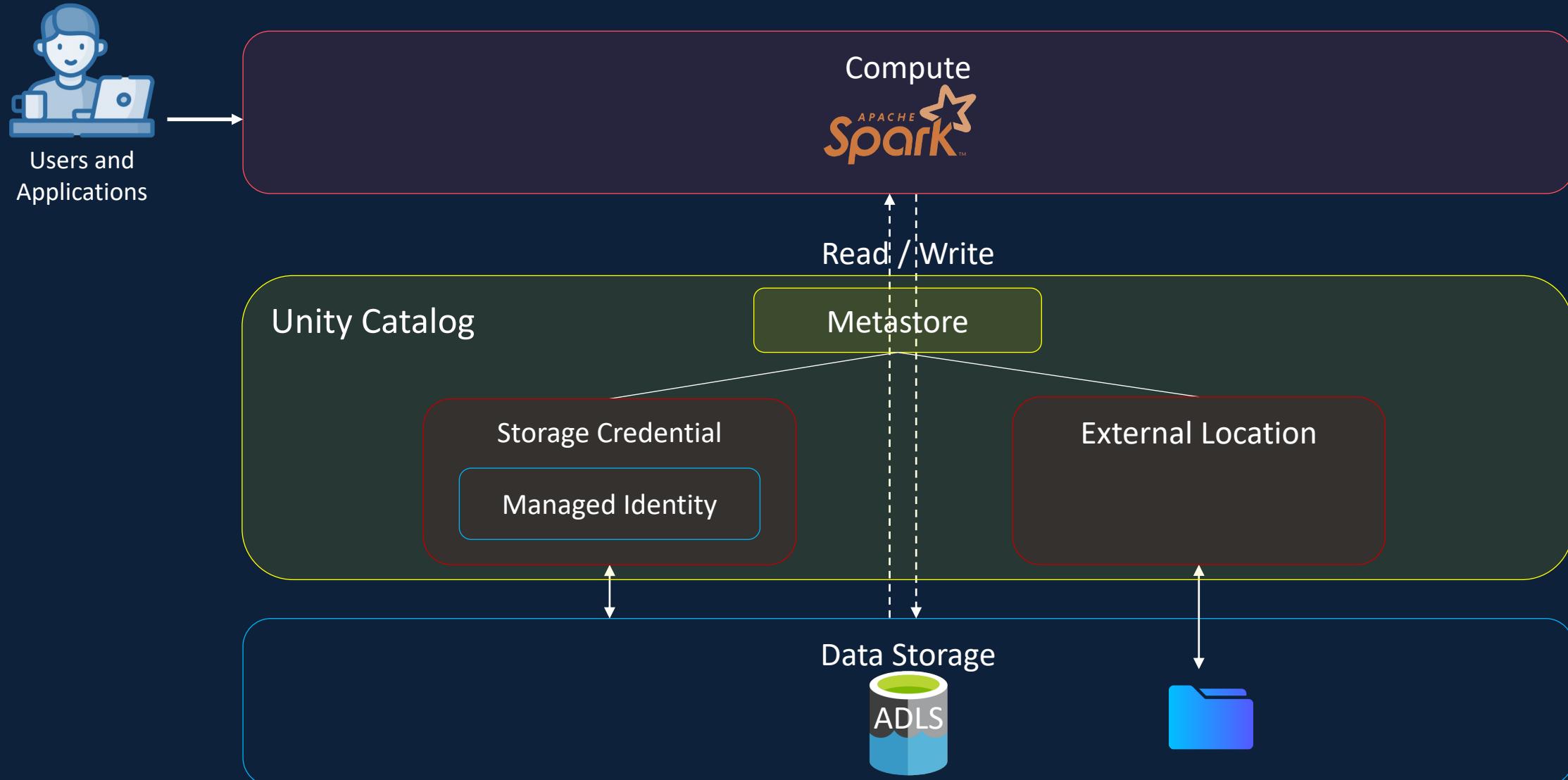
Configure Access to Cloud Storage



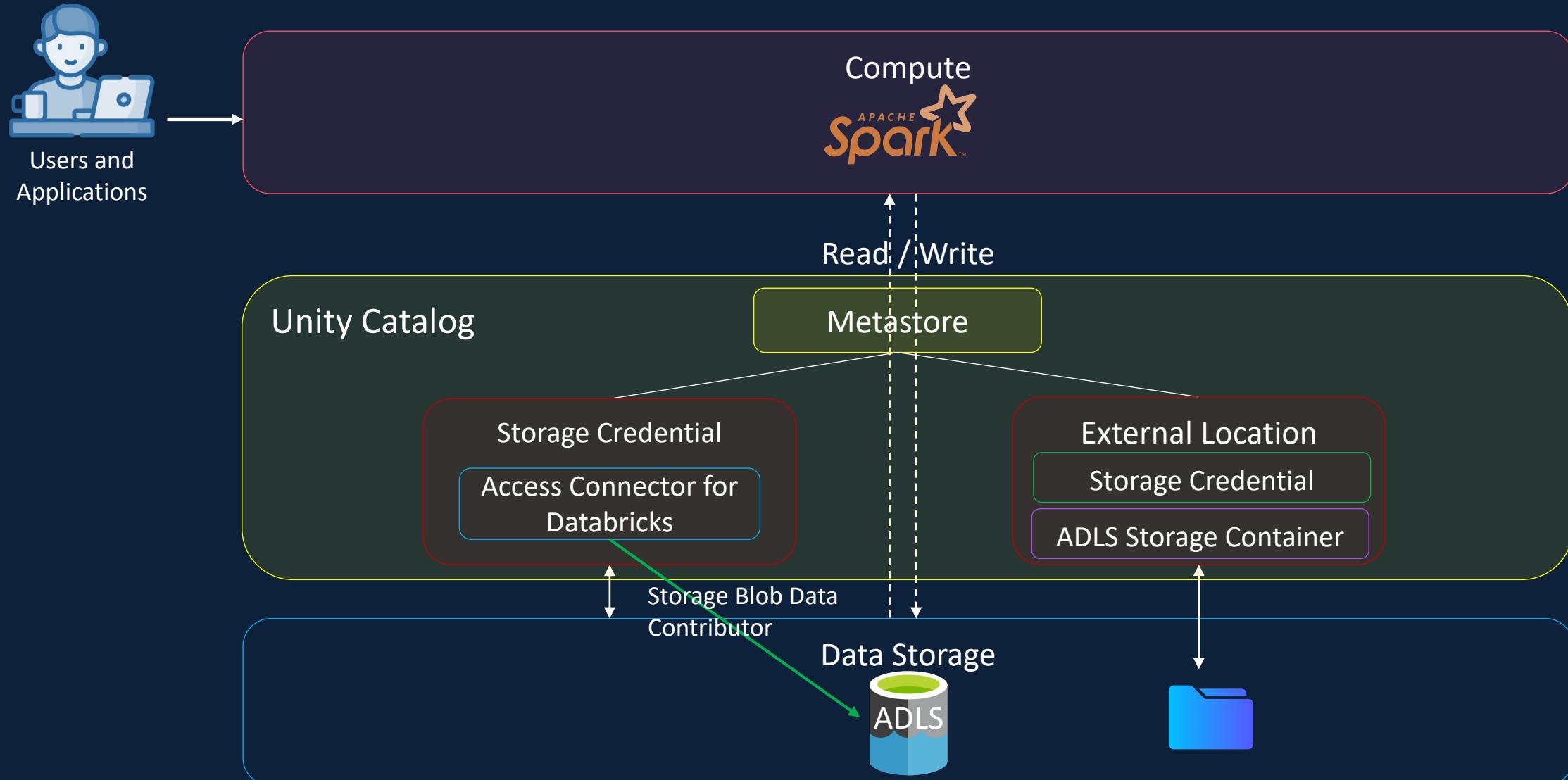
Configure Access to Cloud Storage



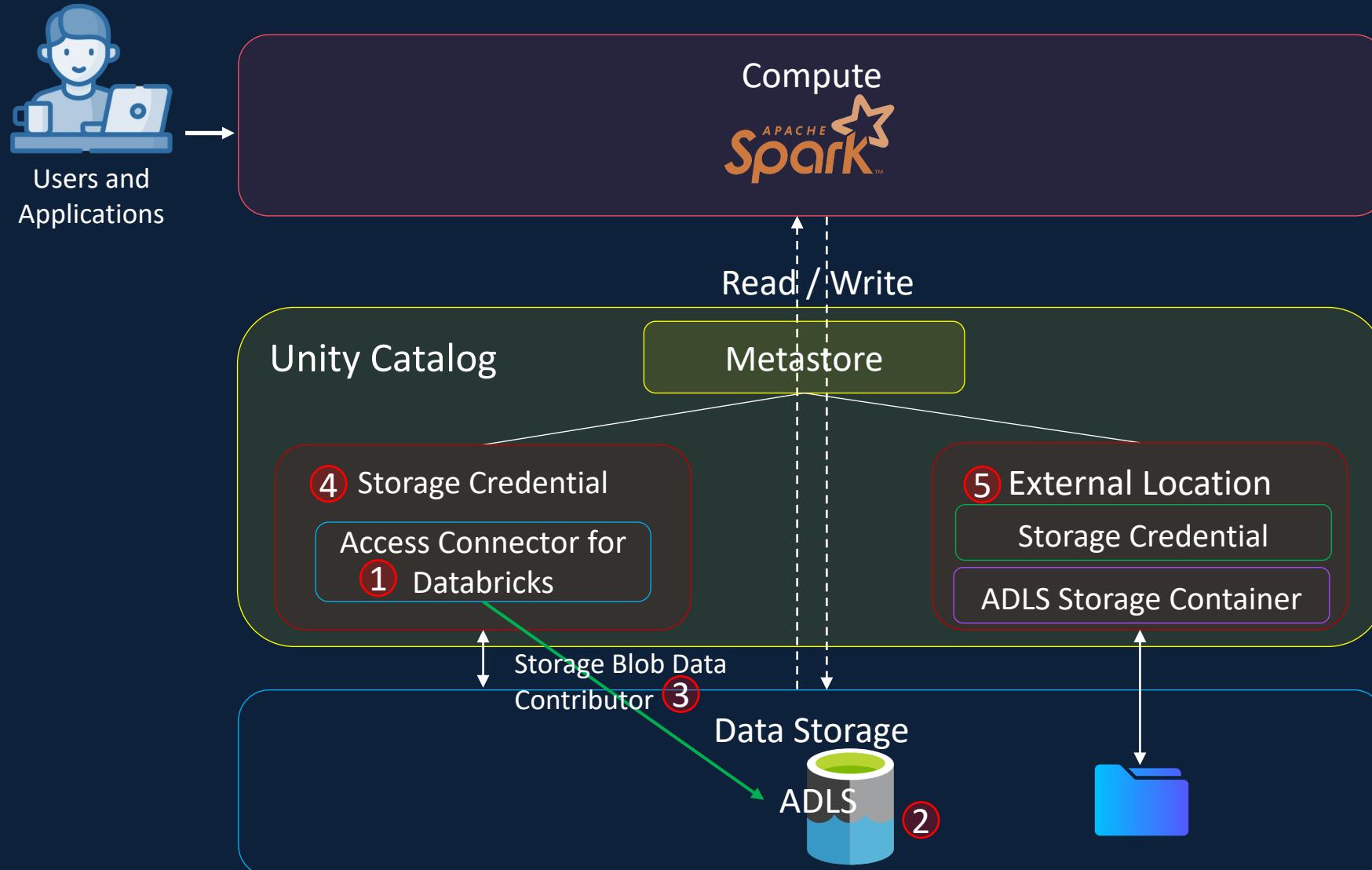
Configure Access to Cloud Storage



Configure Access to Cloud Storage



Configure Access to Cloud Storage

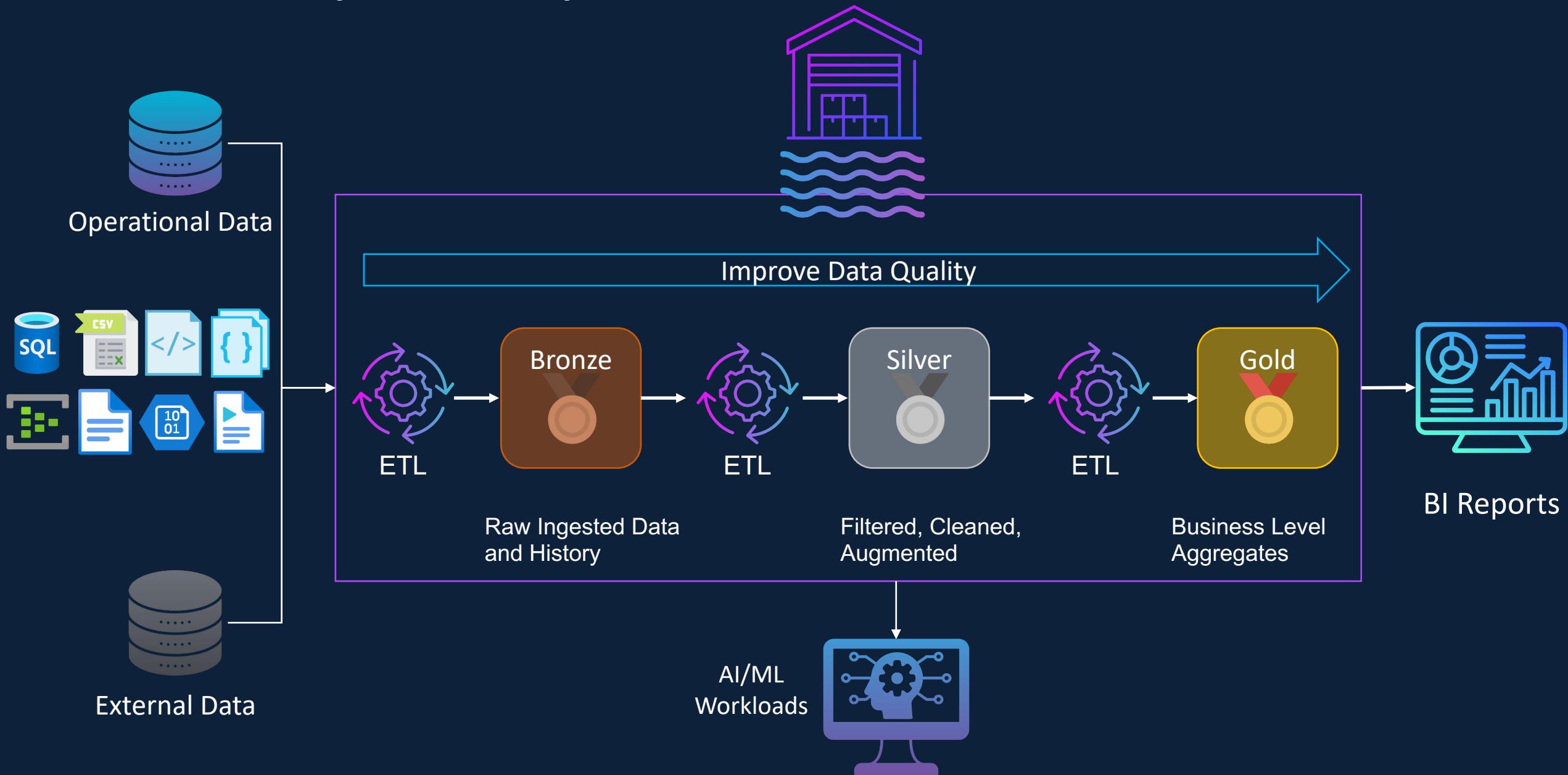


1. Create Access Connector
2. Create Azure Data Lake Storage
3. Assign Storage Blob Data Contributor role
4. Create Storage Credential
5. Create External Location

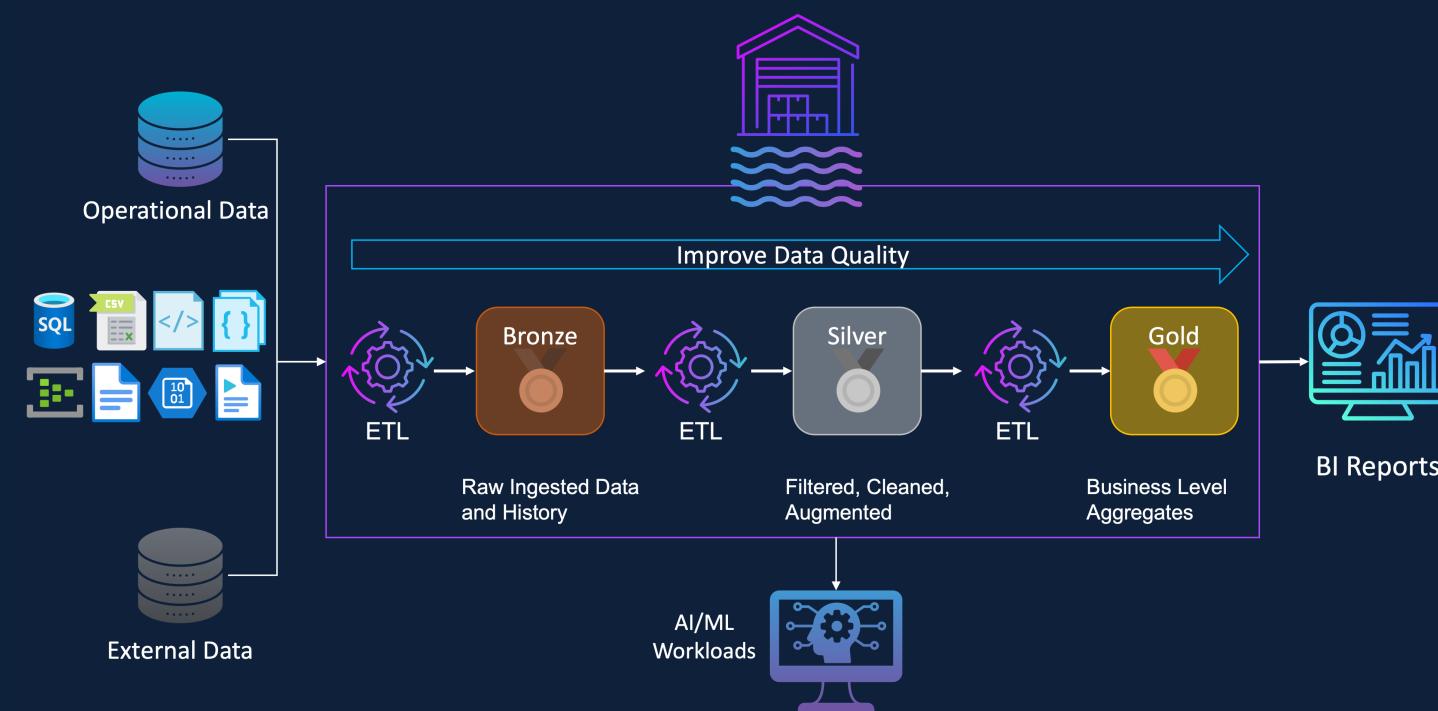
ETL With Apache Spark

Introduction

ETL With Apache Spark



ETL With Apache Spark



Query Files / Folders

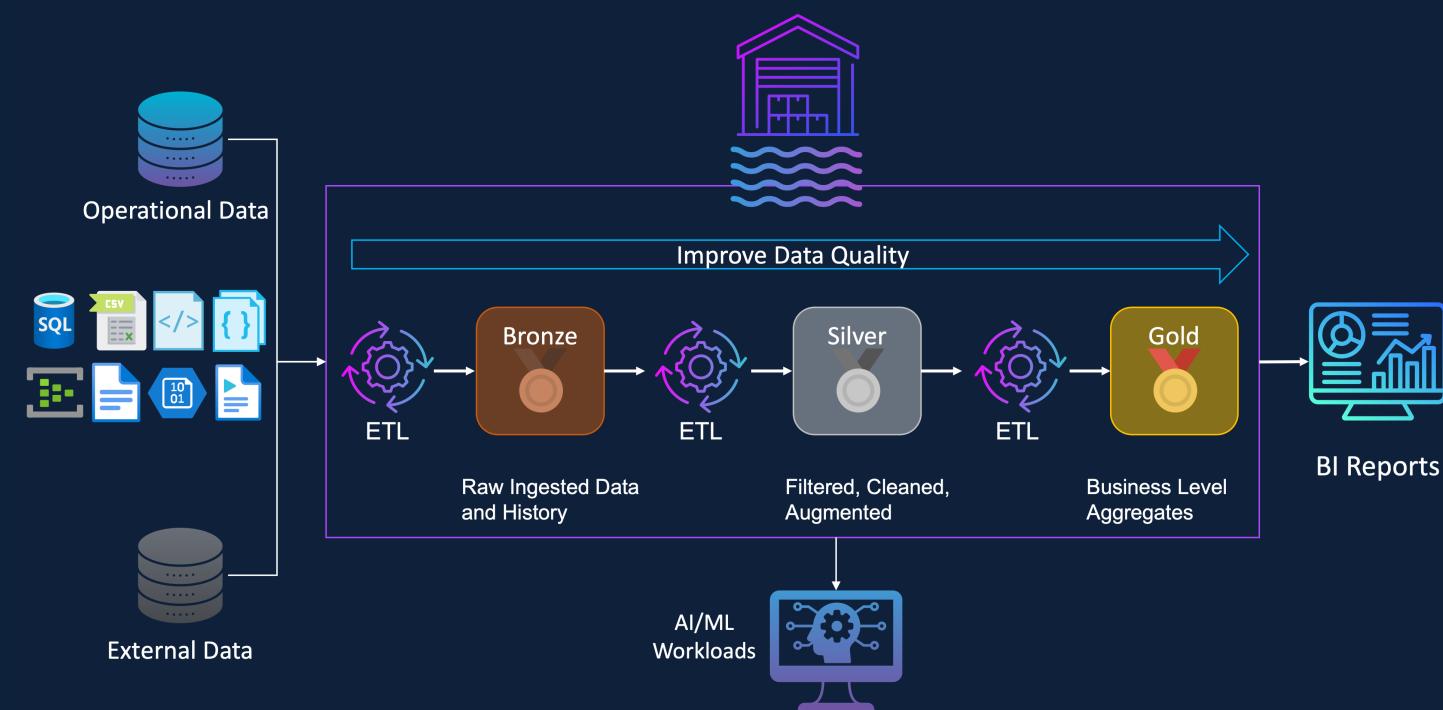
Query External Tables (Azure SQL, MySQL)

Validate and Clean the Data

Transform Data

Create Business Level Aggregates

ETL With Apache Spark



CSV, JSON and Binary Files

Access External Data via JDBC

Direct Query, External Tables, Views

Simple & Complex JSON Structures

Validate and Fix Data Issues

Simple Transformations

User Defined Functions

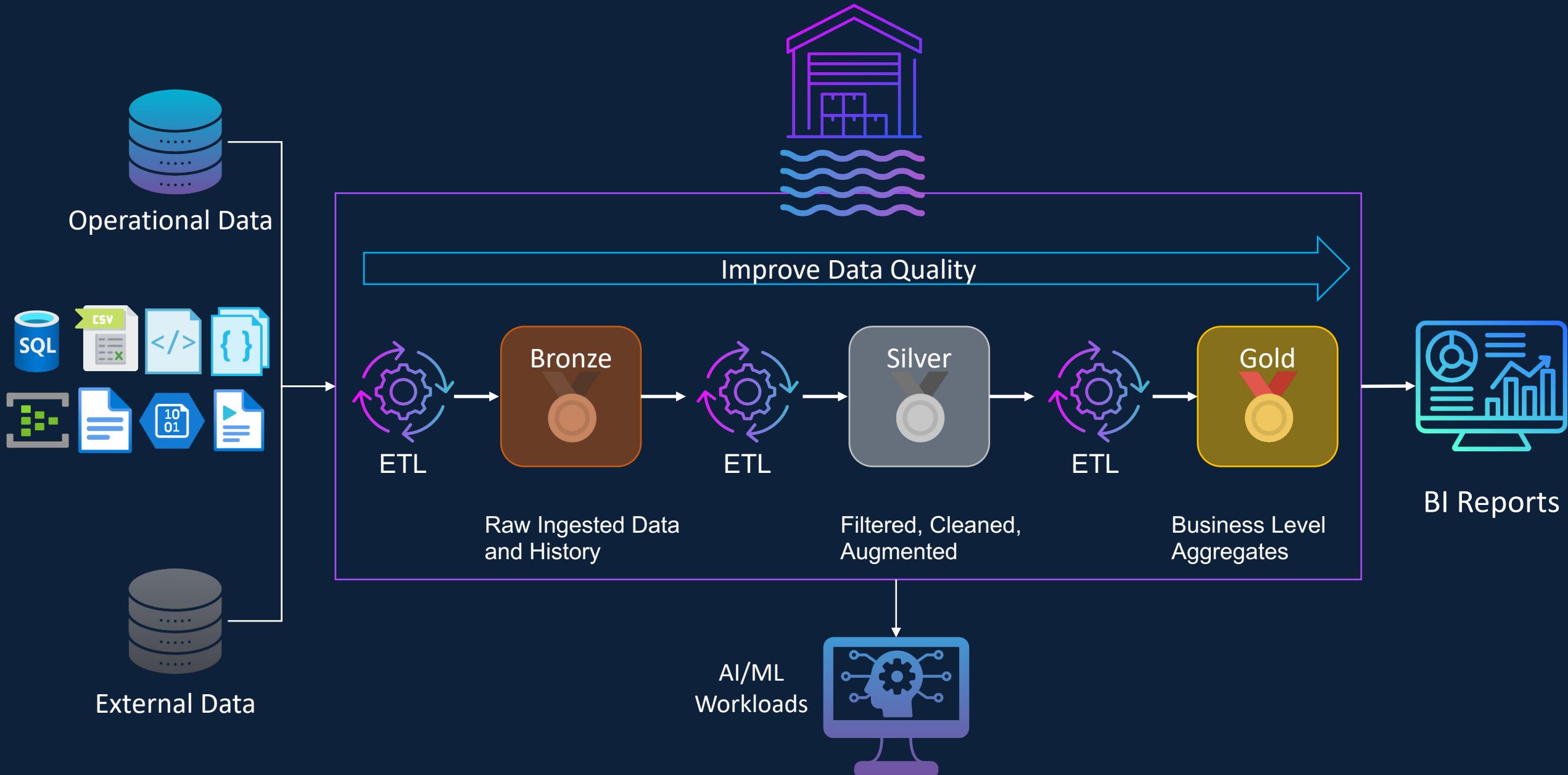
Create Tables

Spark SQL & Python

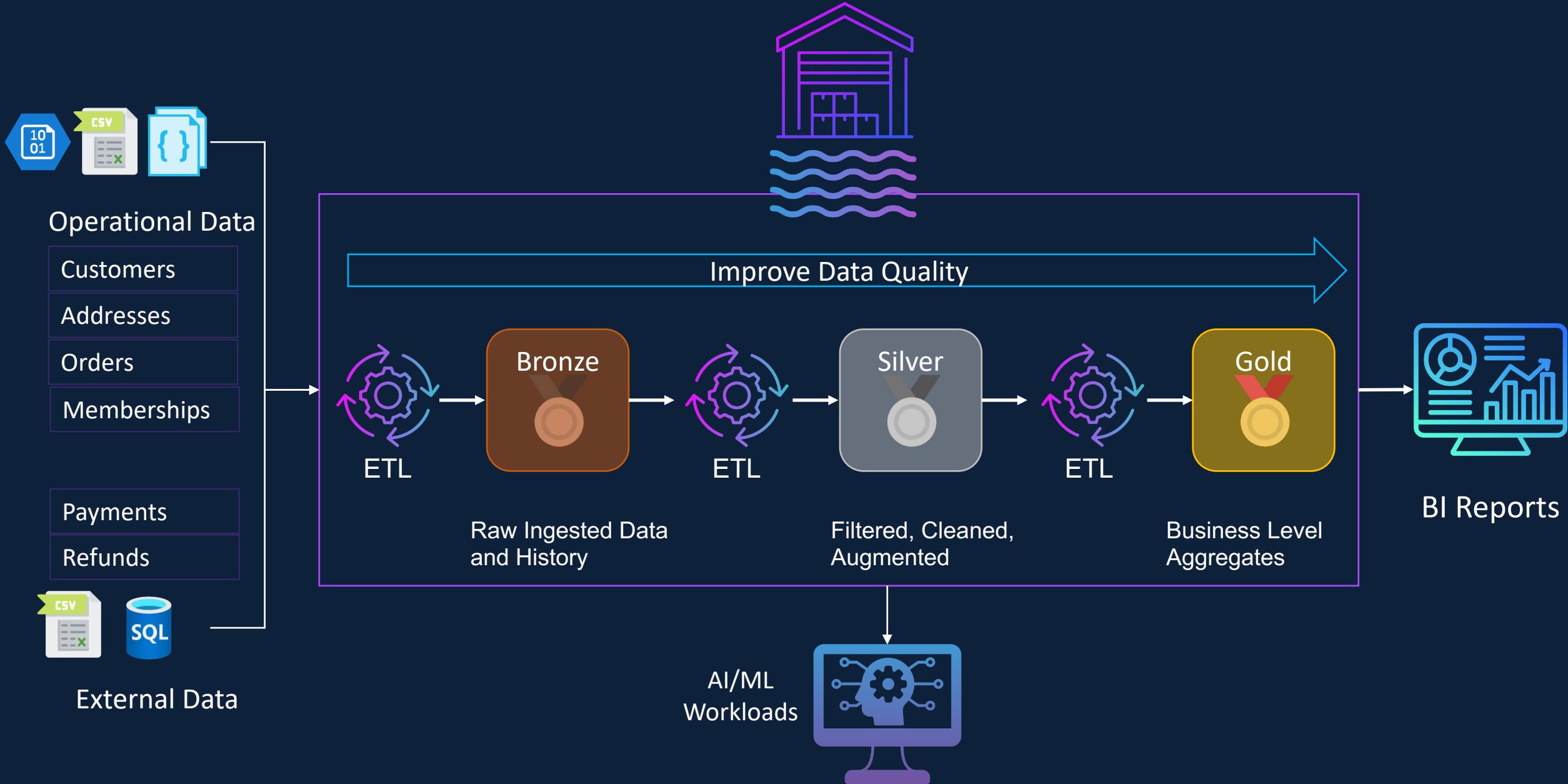
ETL With Apache Spark

Project Introduction

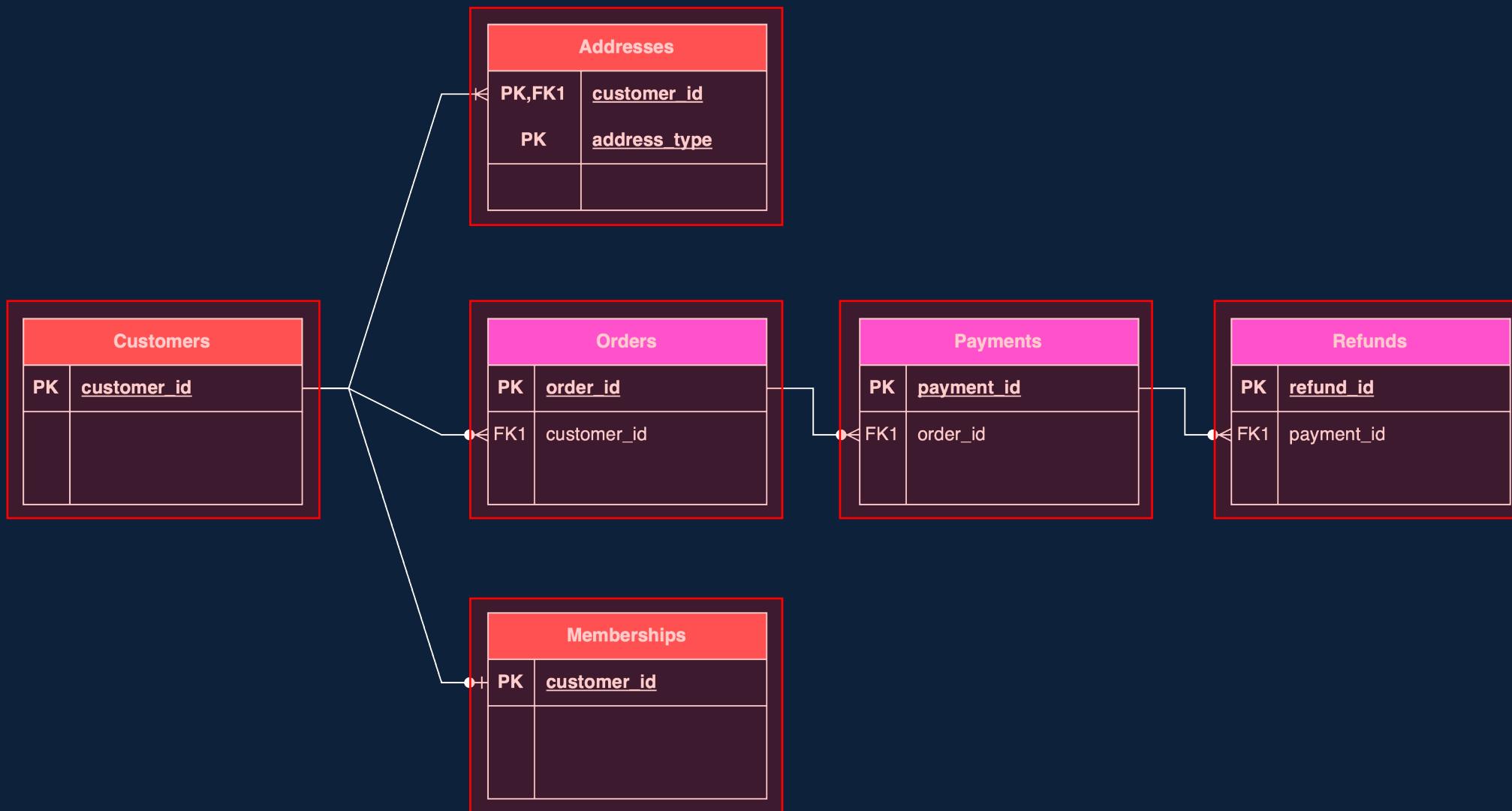
Data Lakehouse Project - GizmoBox



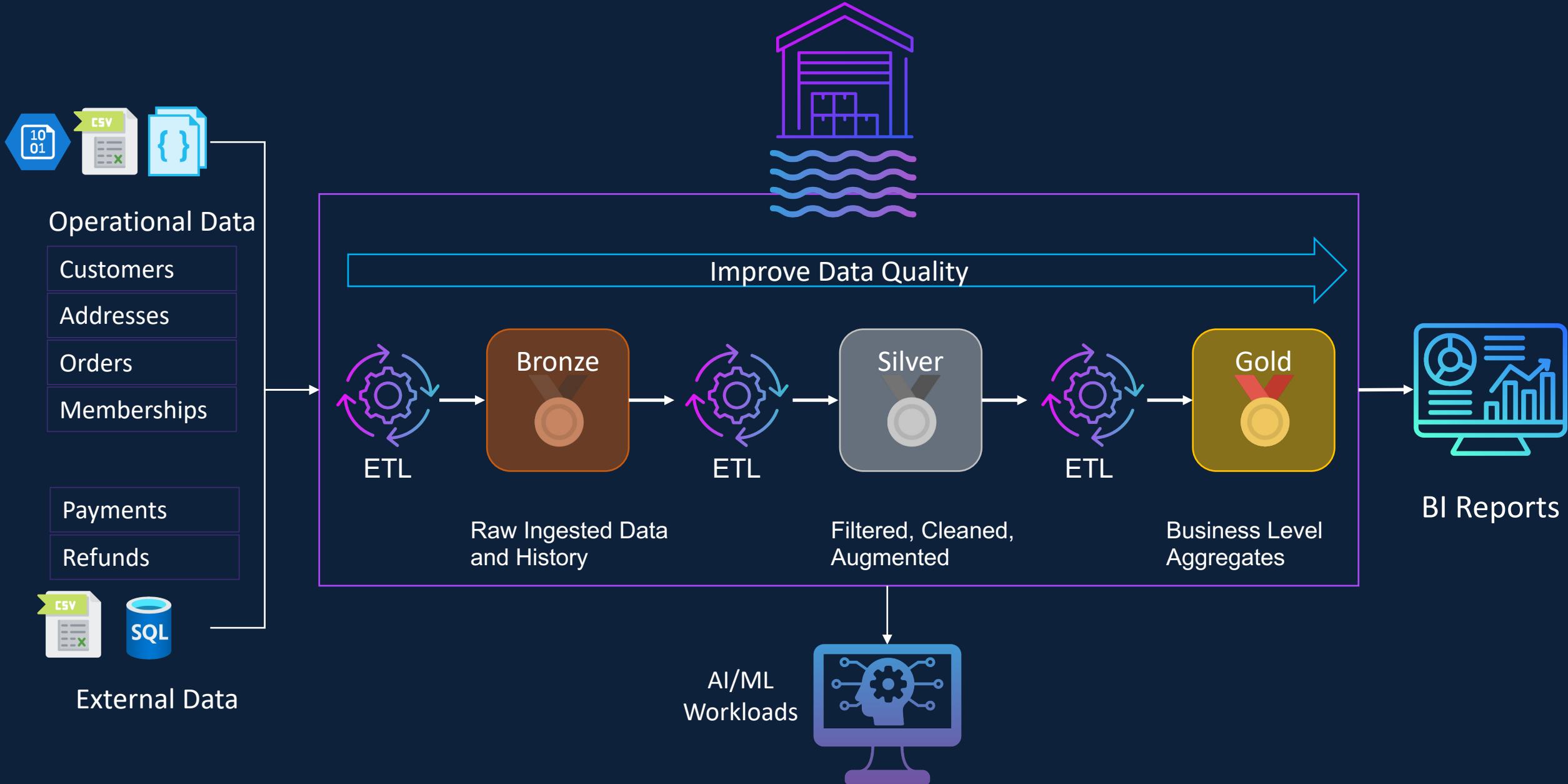
Data Lakehouse Project - GizmoBox



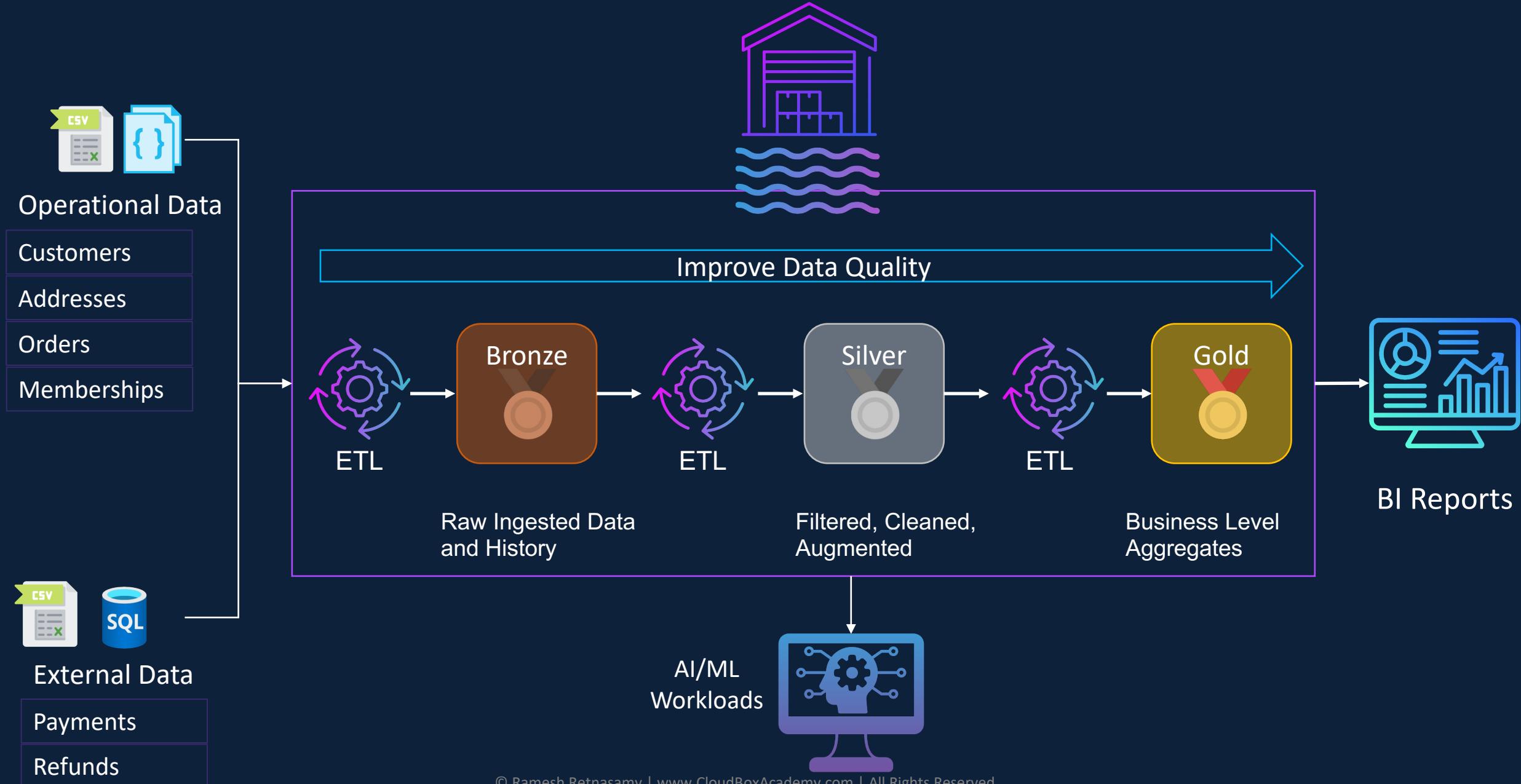
Data Lakehouse Project - GizmoBox



Data Lakehouse Project - GizmoBox



Data Architecture - GizmoBox



Data Architecture - GizmoBox



Landing



Bronze



Silver



Gold

Operational Data

Customers

Addresses

Orders

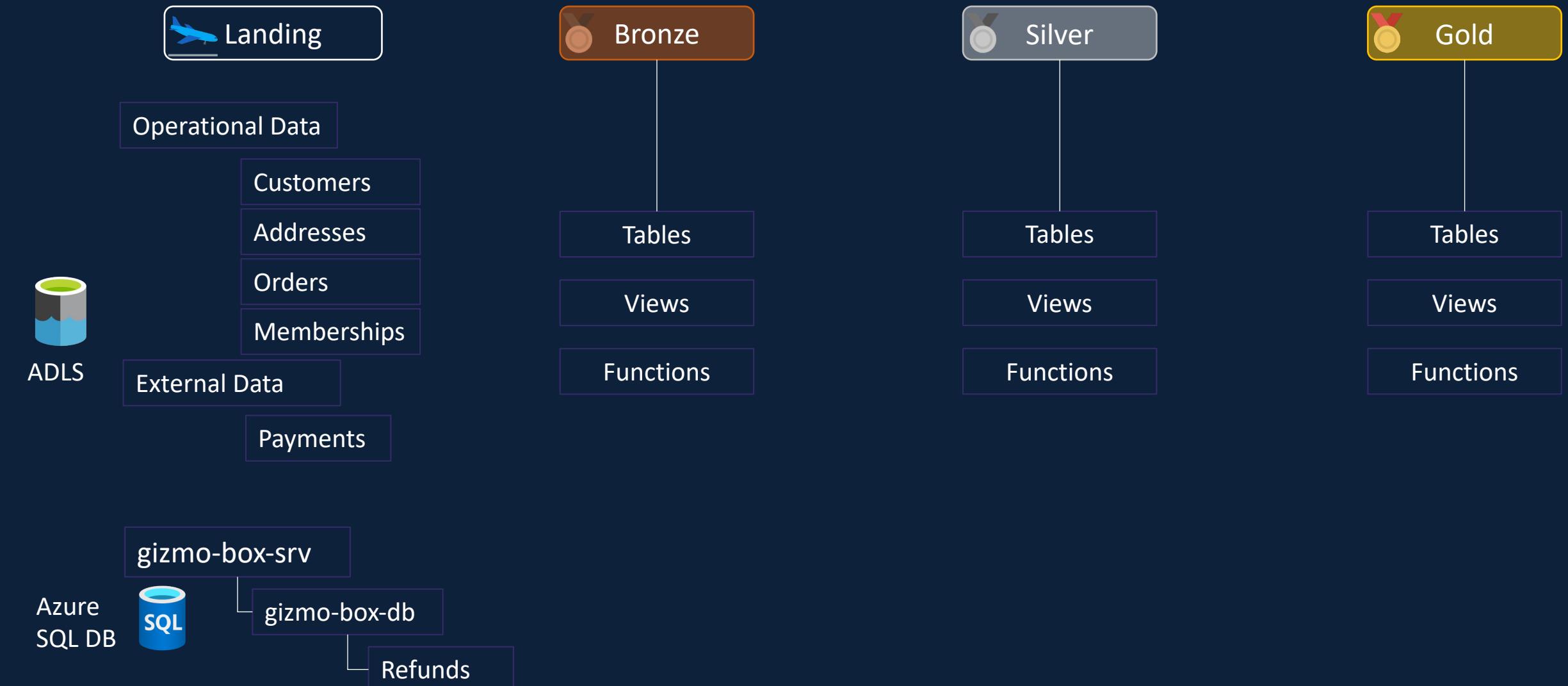
Memberships

External Data

Payments

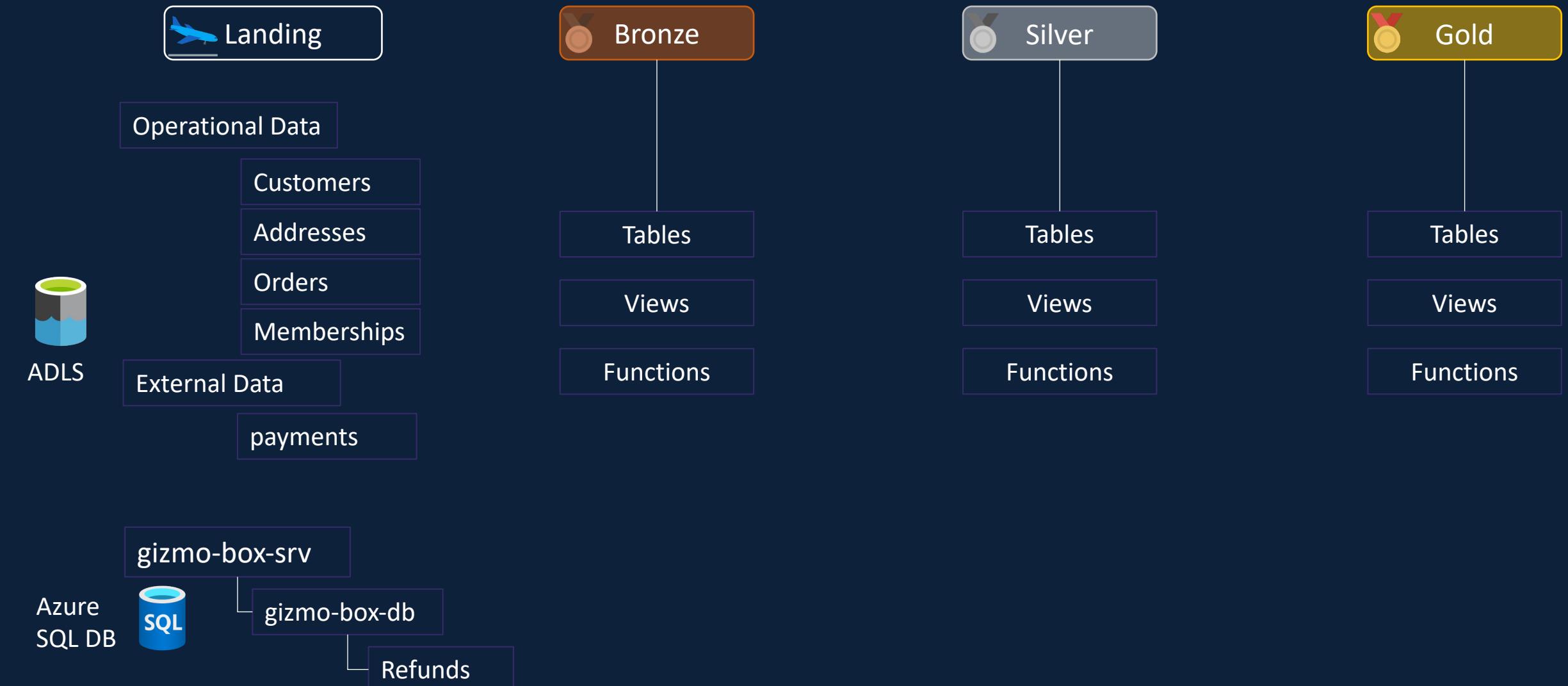
Refunds

Data Architecture - GizmoBox



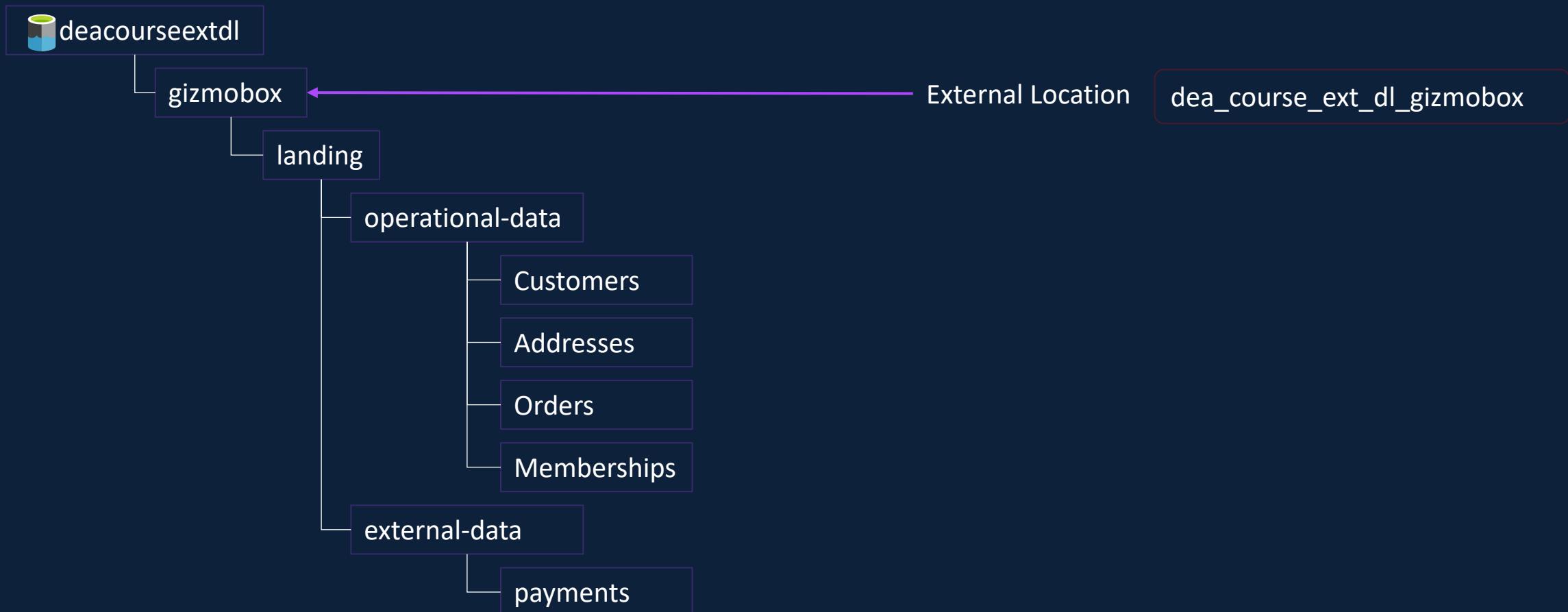
Set-up Project Environment

Data Architecture - GizmoBox



Data Architecture - GizmoBox

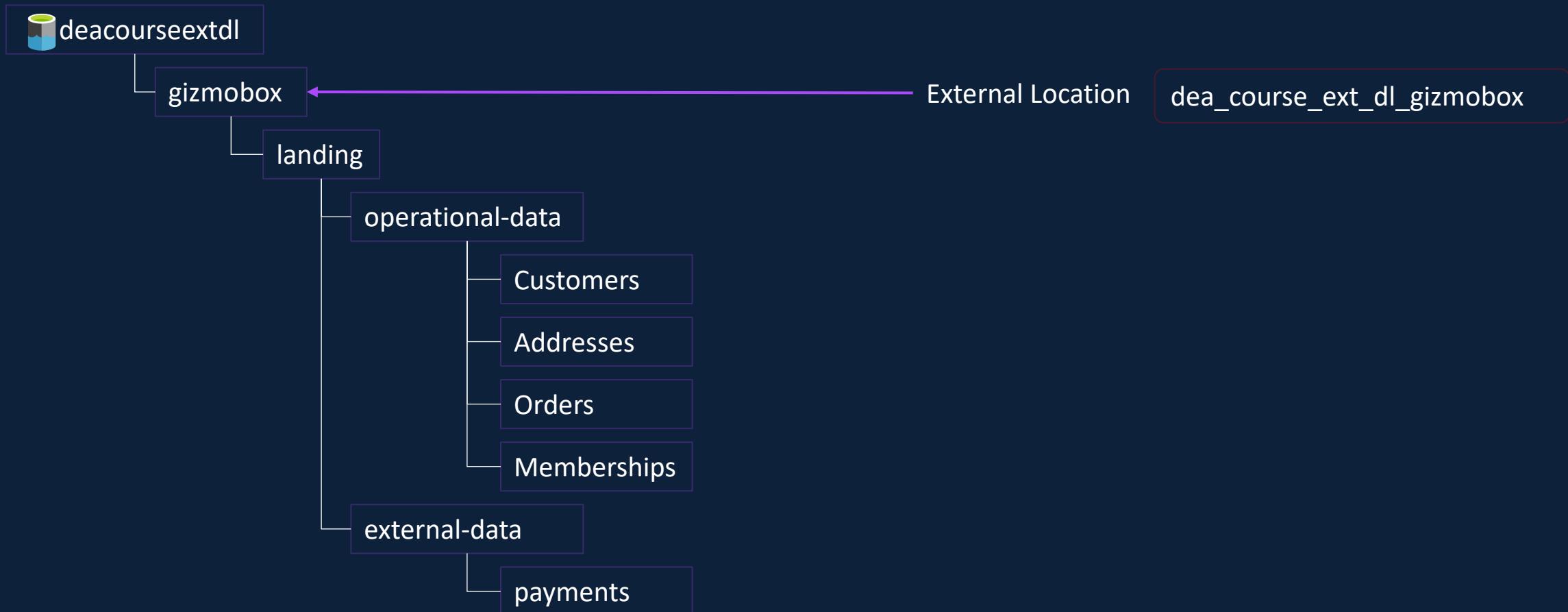
Azure Data Lake Storage Set-Up



Unity Catalog Set-up

Data Architecture - GizmoBox

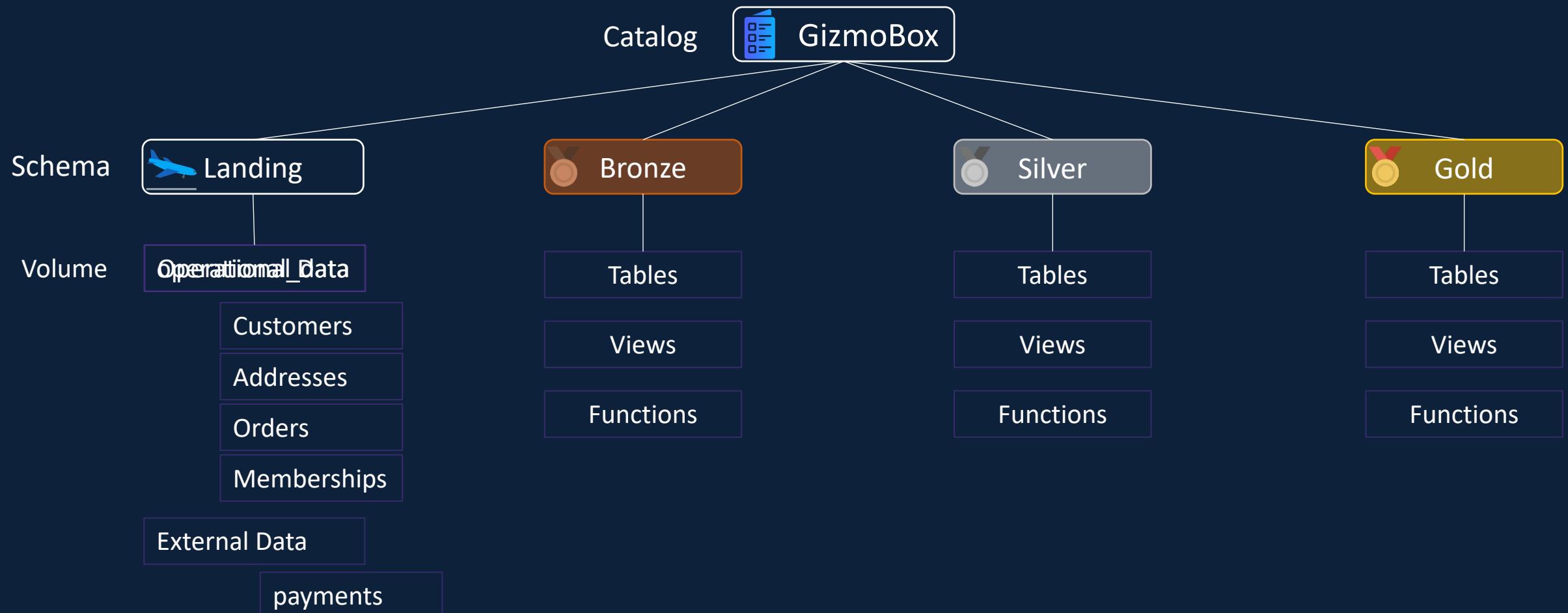
Azure Data Lake Storage Set-Up



Unity Catalog Set-up

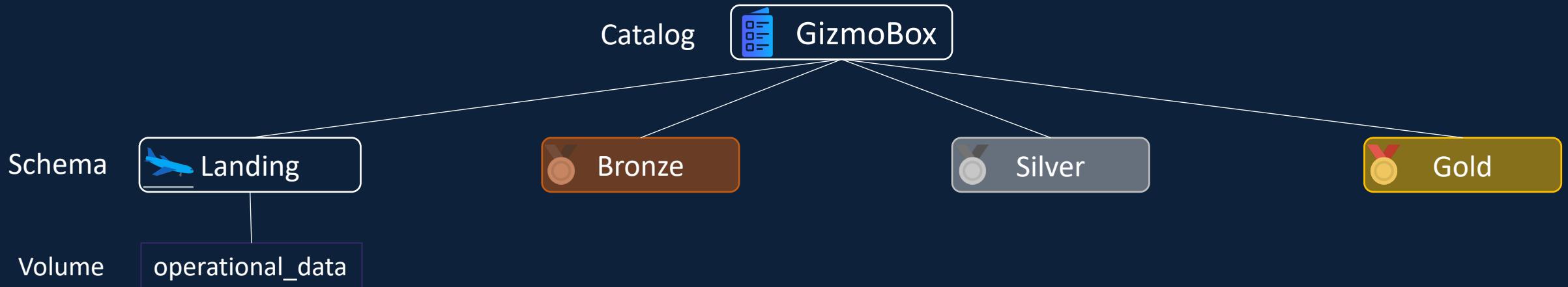
Data Architecture - GizmoBox

Unity Catalog



Data Architecture - GizmoBox

Unity Catalog

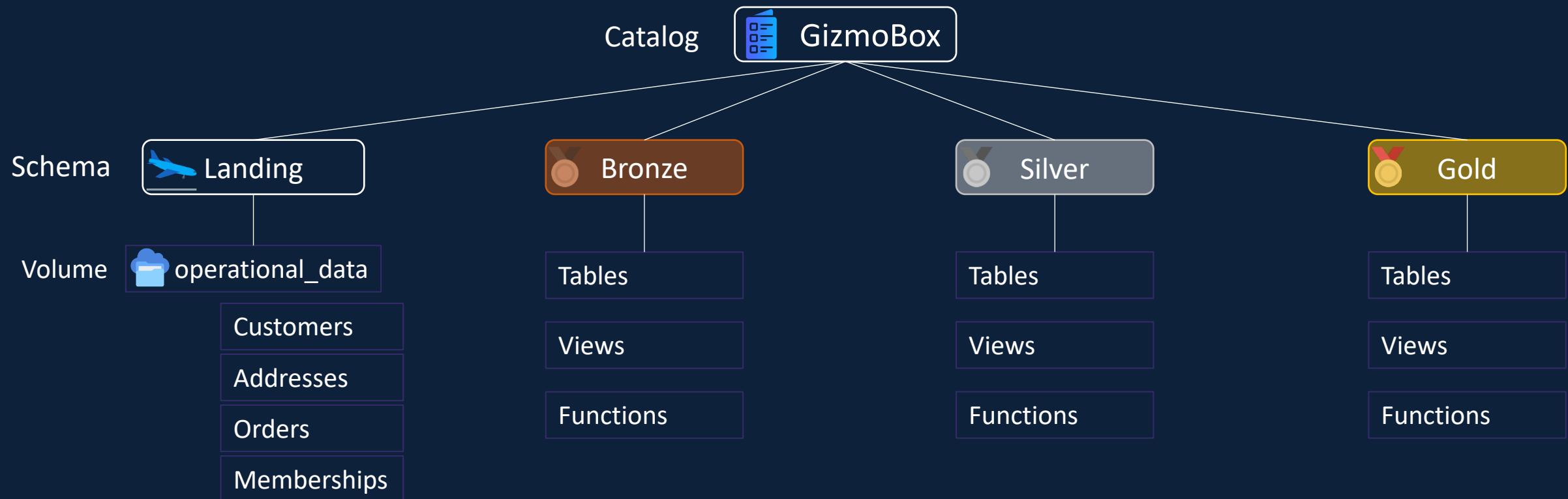


ETL With Apache Spark

Querying Files

Querying Files

Unity Catalog



Querying Files



Customers

customers_2024_10.json
customers_2024_11.json
customers_2024_12.json
customers_2025_01.json

`SELECT * FROM <file_format>.`<file path>``

CSV

JSON

XML

Parquet

Avro

ORC

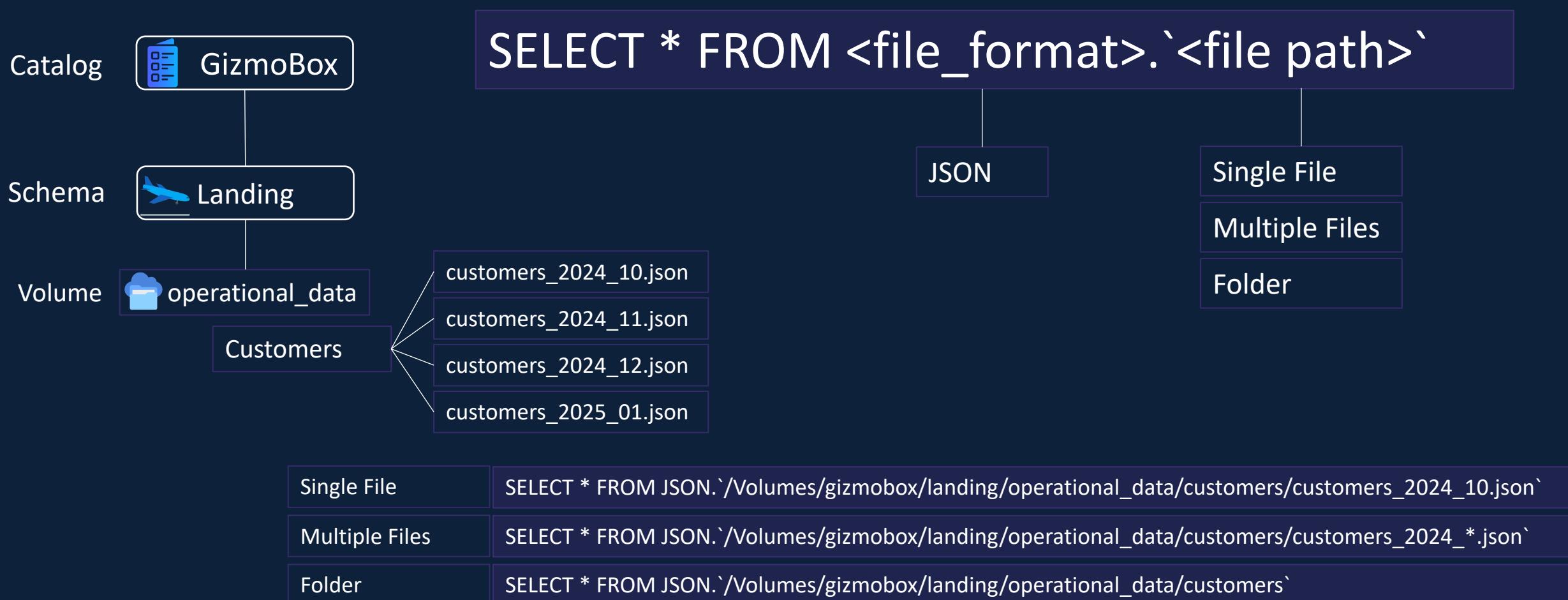
Delta

Text

Binary

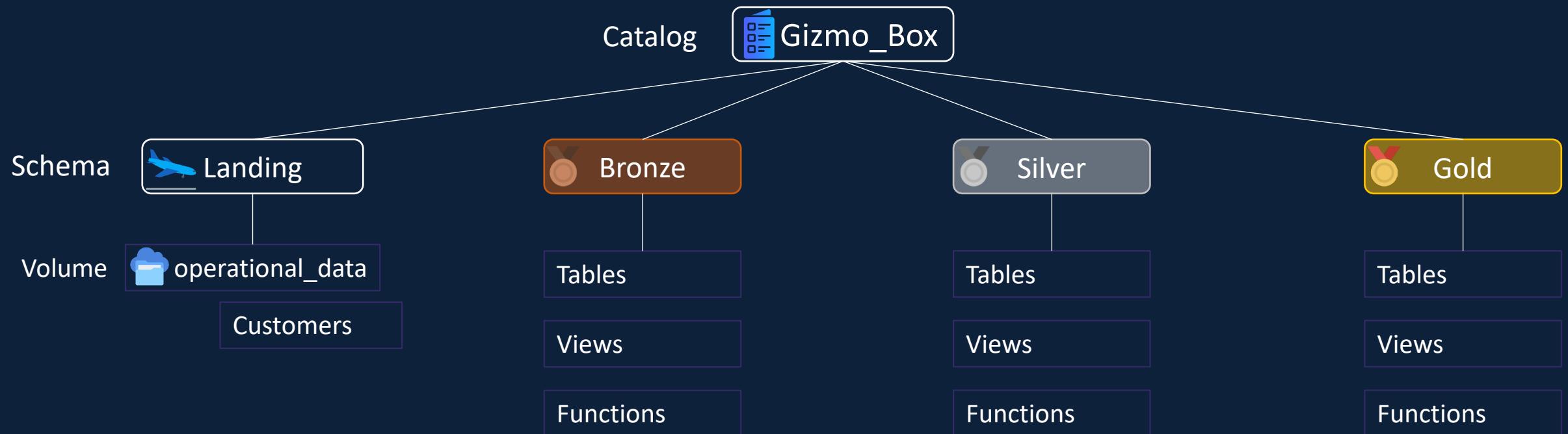
**Backtick
Character**

Querying Files - JSON



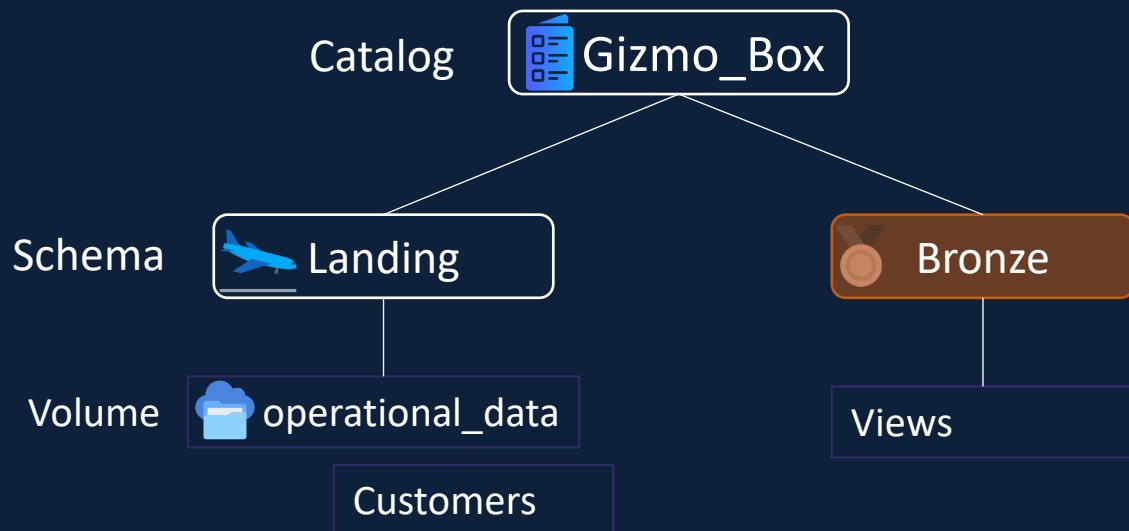
Create Views

Unity Catalog



Create Views

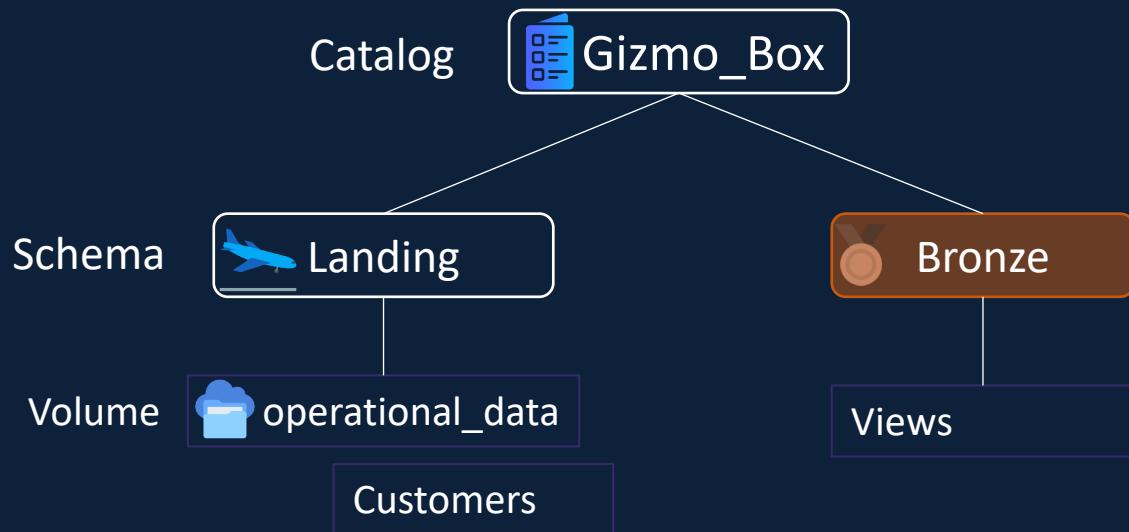
Unity Catalog



```
CREATE OR REPLACE VIEW v_customers
AS
SELECT *
FROM JSON.`/Volumes/gizmo_box/landing/operational_data/customers`
```

Create Views

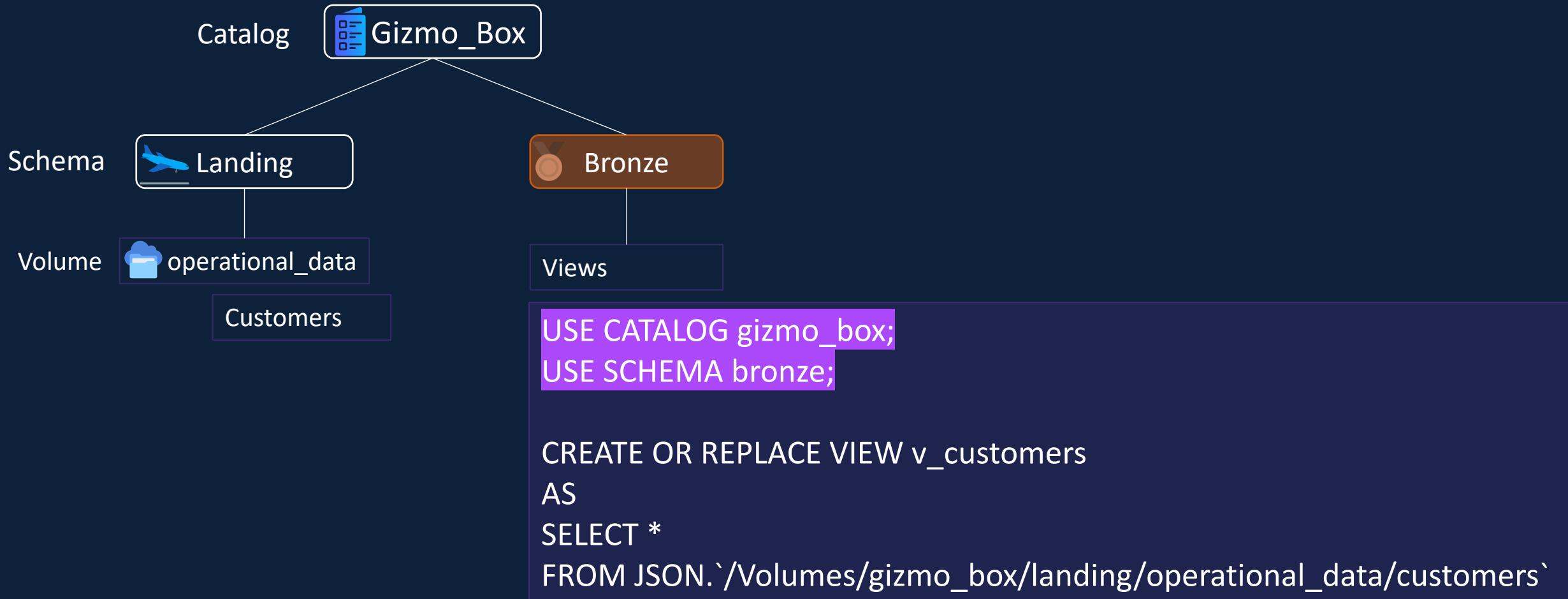
Unity Catalog



```
CREATE OR REPLACE VIEW gizmo_box.bronze.v_customers
AS
SELECT *
FROM JSON.`/Volumes/gizmo_box/landing/operational_data/customers`
```

Create Views

Unity Catalog



Create Temporary Views

View

```
CREATE OR REPLACE VIEW tv_customers
AS
SELECT *
FROM JSON.`/Volumes/gizmo_box/landing/operational_data/customers`
```

Temporary View

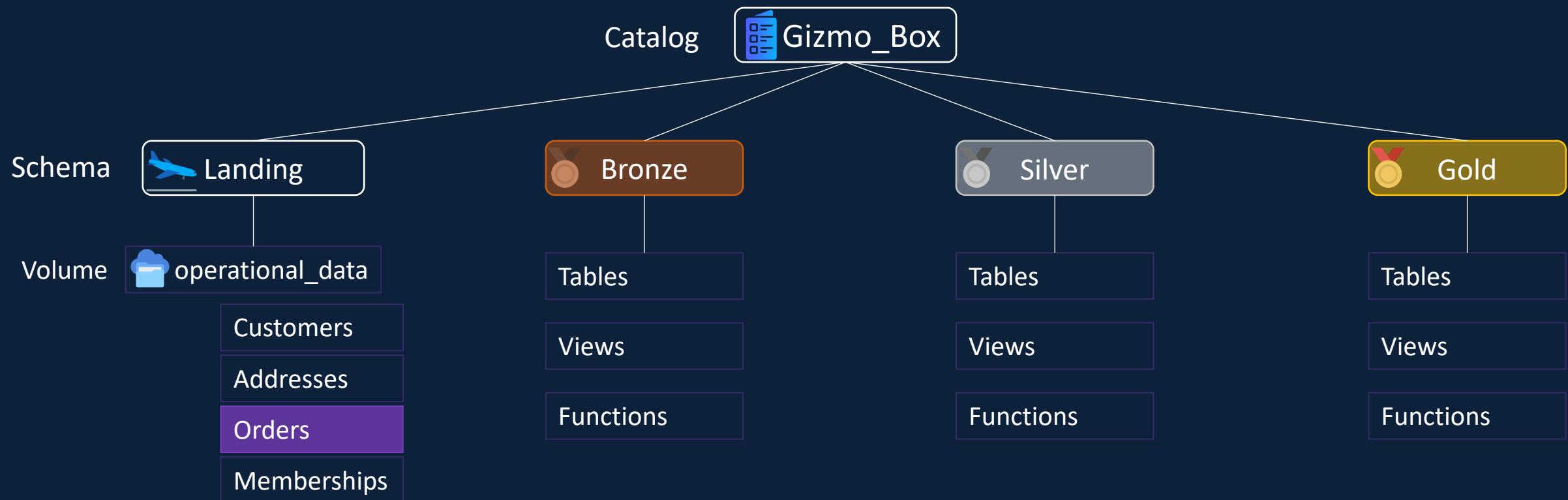
```
CREATE OR REPLACE TEMPORARY VIEW tv_customers
AS
SELECT *
FROM JSON.`/Volumes/gizmo_box/landing/operational_data/customers`
```

Global Temporary View

```
CREATE OR REPLACE GLOBAL TEMPORARY VIEW tv_customers
AS
SELECT *
FROM JSON.`/Volumes/gizmo_box/landing/operational_data/customers`
```

Query Files

Unity Catalog



Query Files

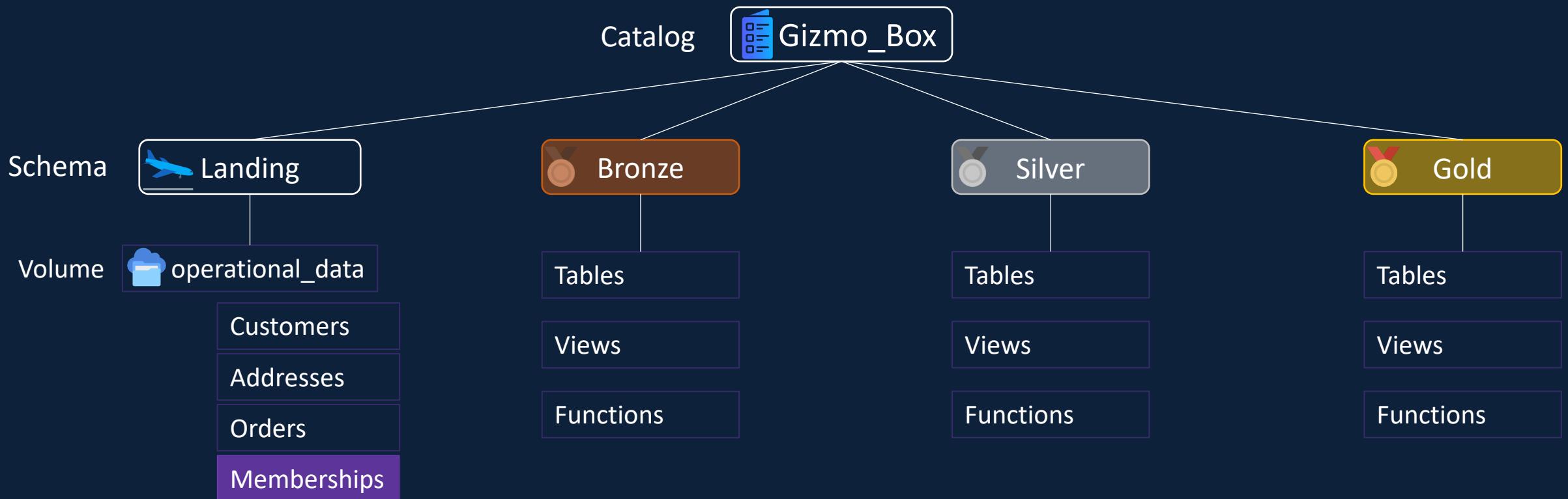


Query Files – JSON/ Text

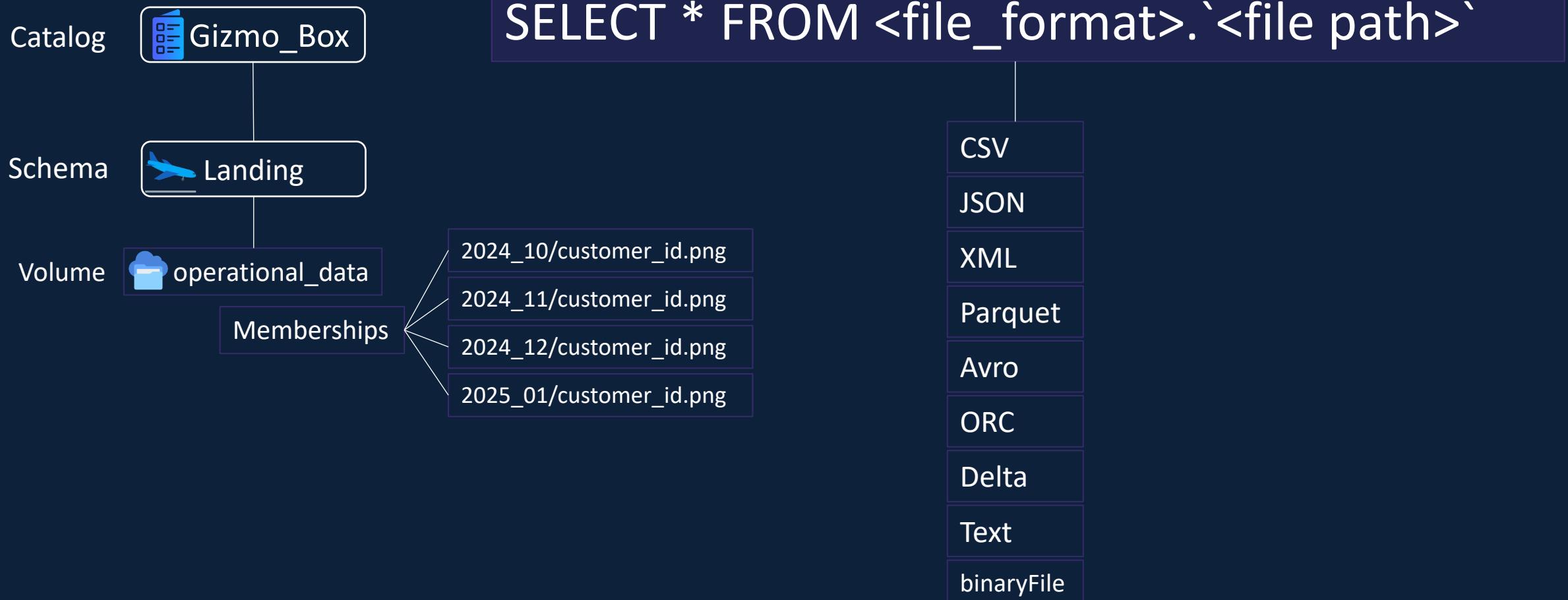


Query Image Files

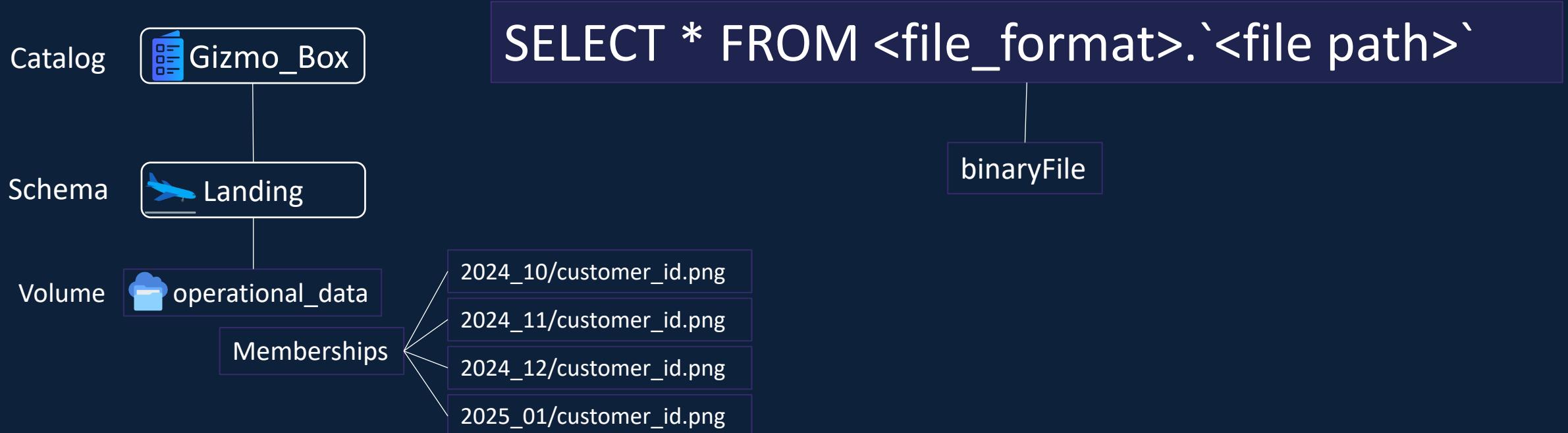
Unity Catalog



Query Image Files

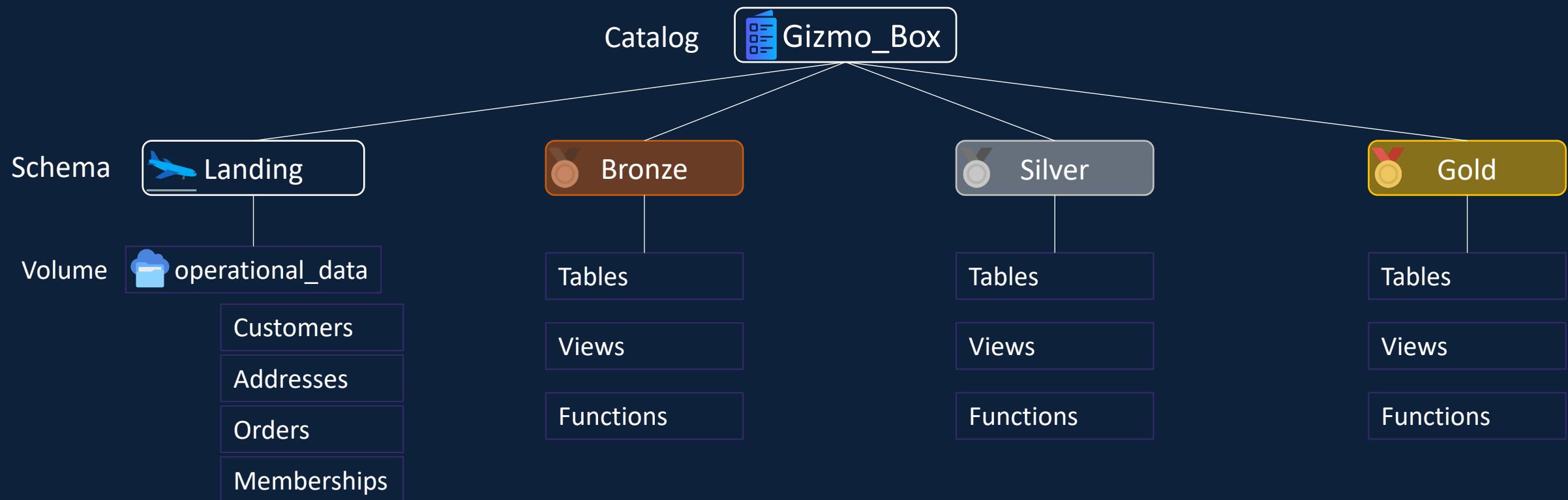


Query Image Files



Query Files – CSV

Unity Catalog



Query Files – CSV



Query Files – CSV



Query Files – CSV

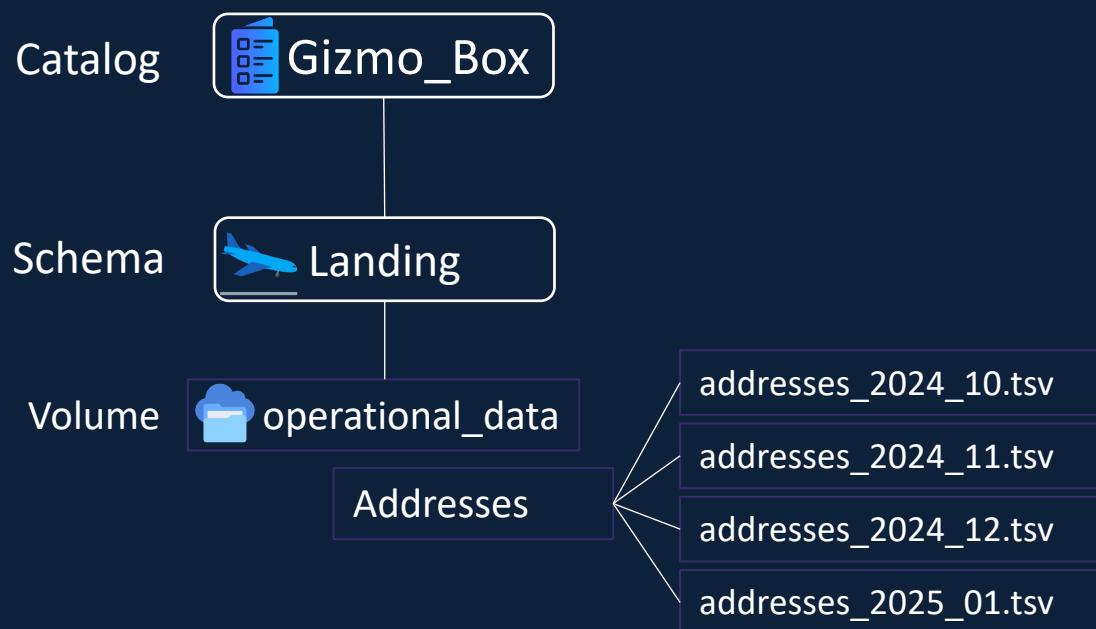


Table Valued Functions (e.g. read_files)

External Tables

Create External Tables

Azure Data Lake Storage



Create External Tables

Azure Data Lake Storage

Databricks Unity Catalog

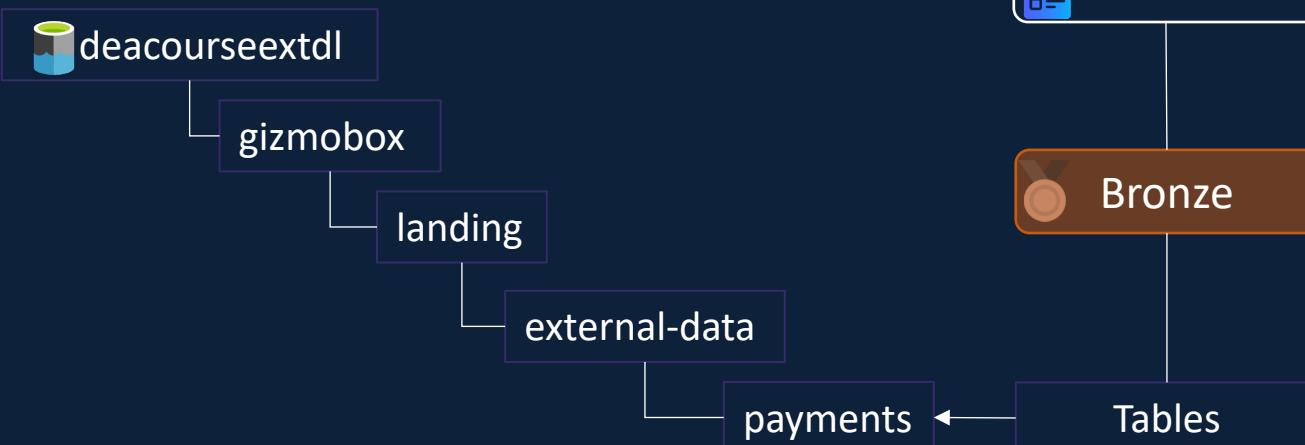


```
CREATE TABLE <catalog_name.schema_name.table_name>
(
    List of column_names column_types
)
USING <file format>
OPTIONS (
    List of key = value
)
LOCATION "file_path";
```

Create External Tables

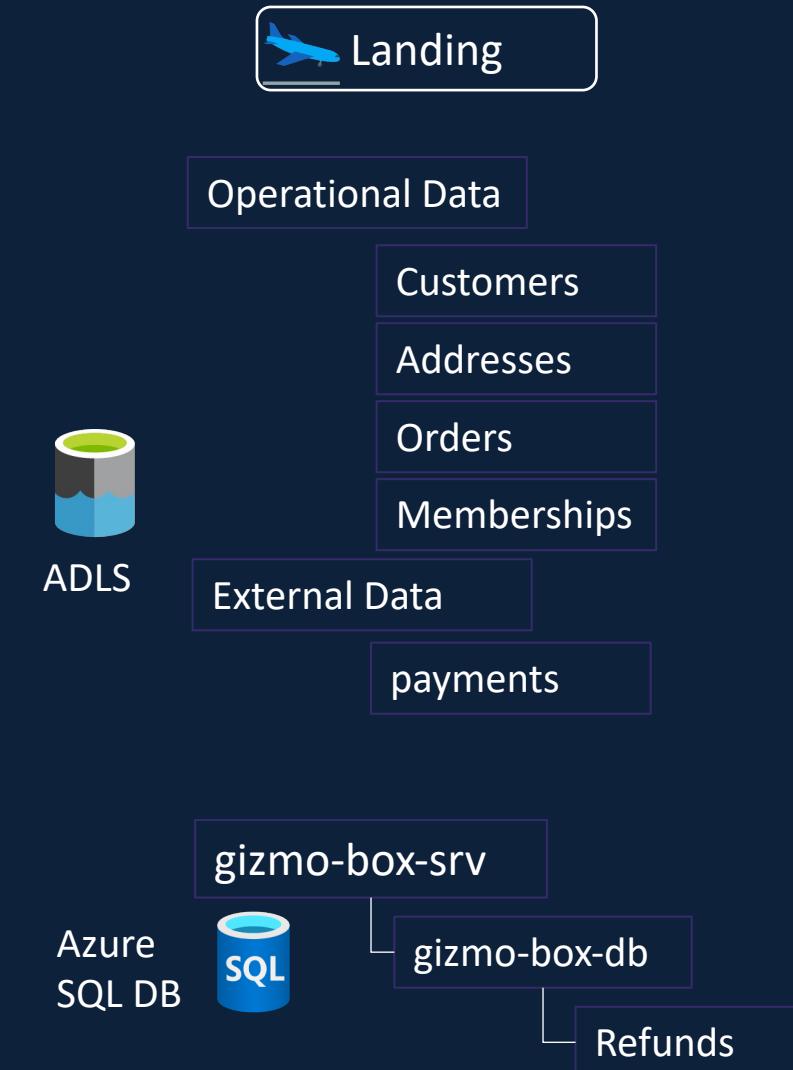
Azure Data Lake Storage

Databricks Unity Catalog

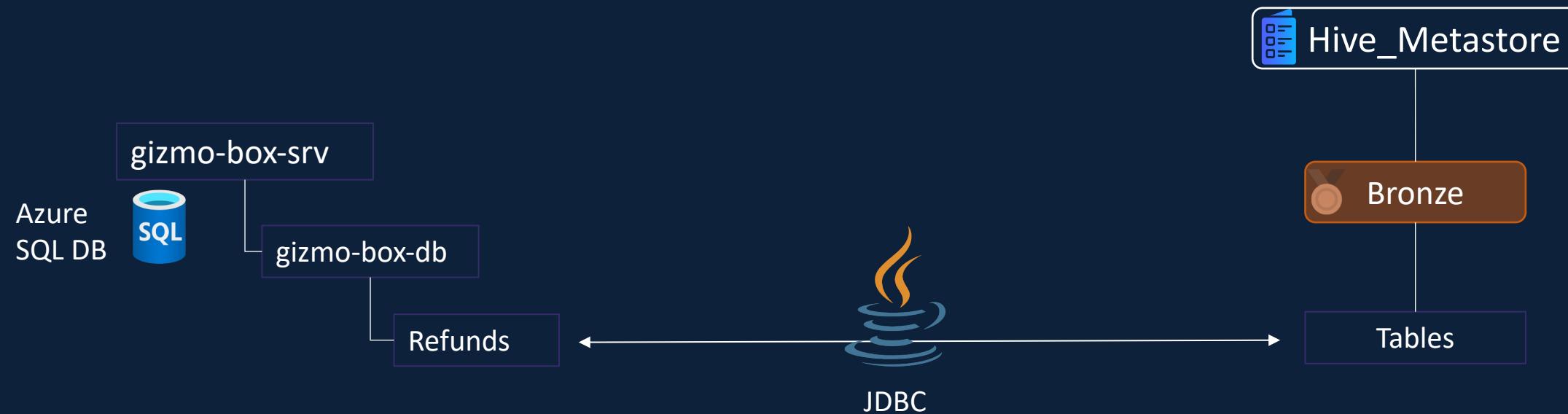


```
CREATE TABLE hive_metastore.bronze.payments
(
    payment_id INTEGER,
    customer_id INTEGER,
    payment_date TIMESTAMP,
    payment_status STRING,
    payment_method STRING
)
USING CSV
OPTIONS (
    header = "true",
    delimiter = ","
)
LOCATION
"abfss://gizmobox@deacourseextdl.dfs.core.windows.net/landing/
external_data/payments";
```

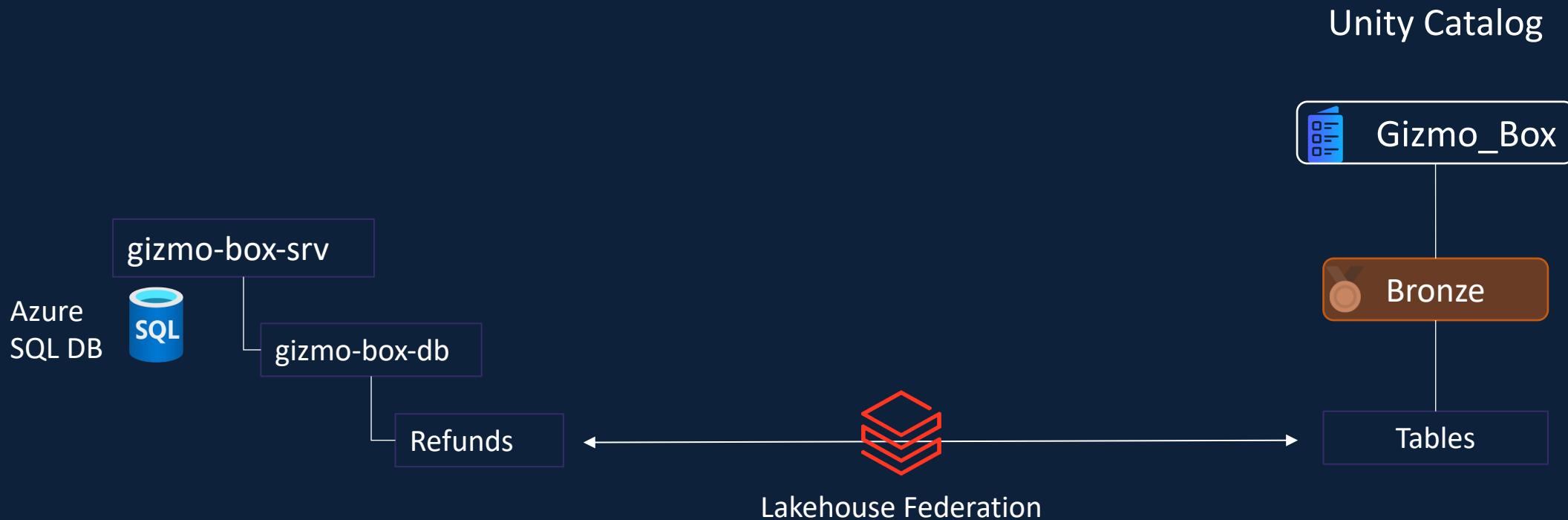
Create External Tables – Via JDBC



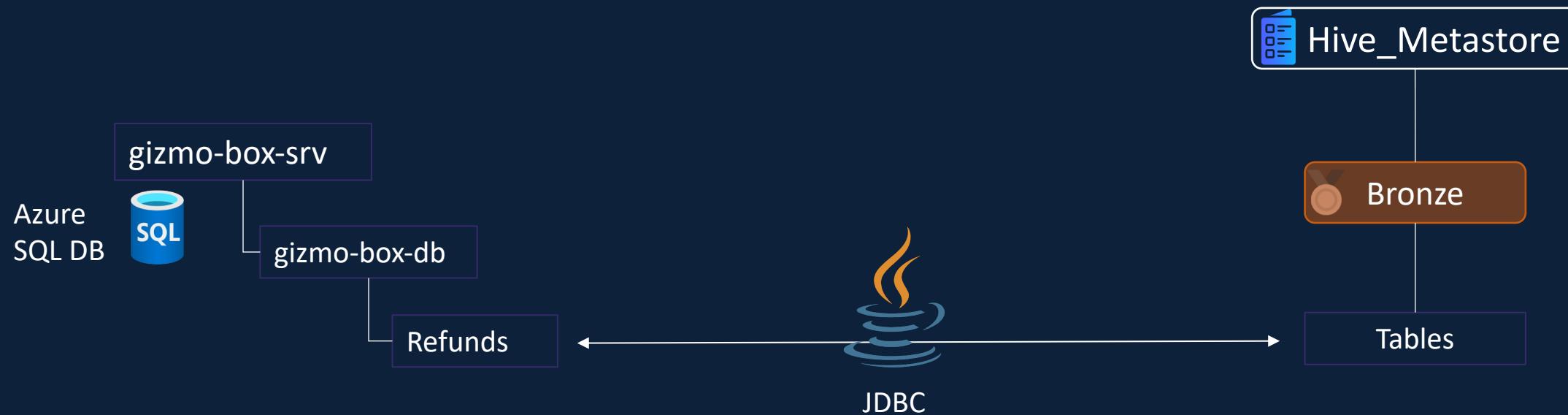
Create External Tables – Via JDBC



Create External Tables – Via JDBC



Create External Tables – Via JDBC



Spark Structured Streaming

Spark Structured Streaming

Batch Processing

Data is collected over a period of time, and then processed in bulk at scheduled intervals.

Insights are delayed

Not suitable for real time analytics

Spark Structured Streaming

Stream Processing

Data is processed continuously as it arrives, enabling real-time and near real-time analytics

Real time fraud detection

Live monitoring

Instant product recommendation

Spark Structured Streaming

Spark Structured Streaming

Scalable, Fault Tolerant Stream Processing Engine built on top of Apache Spark.

Uses Dataframe and Dataset APIs

Micro-Batch Processing

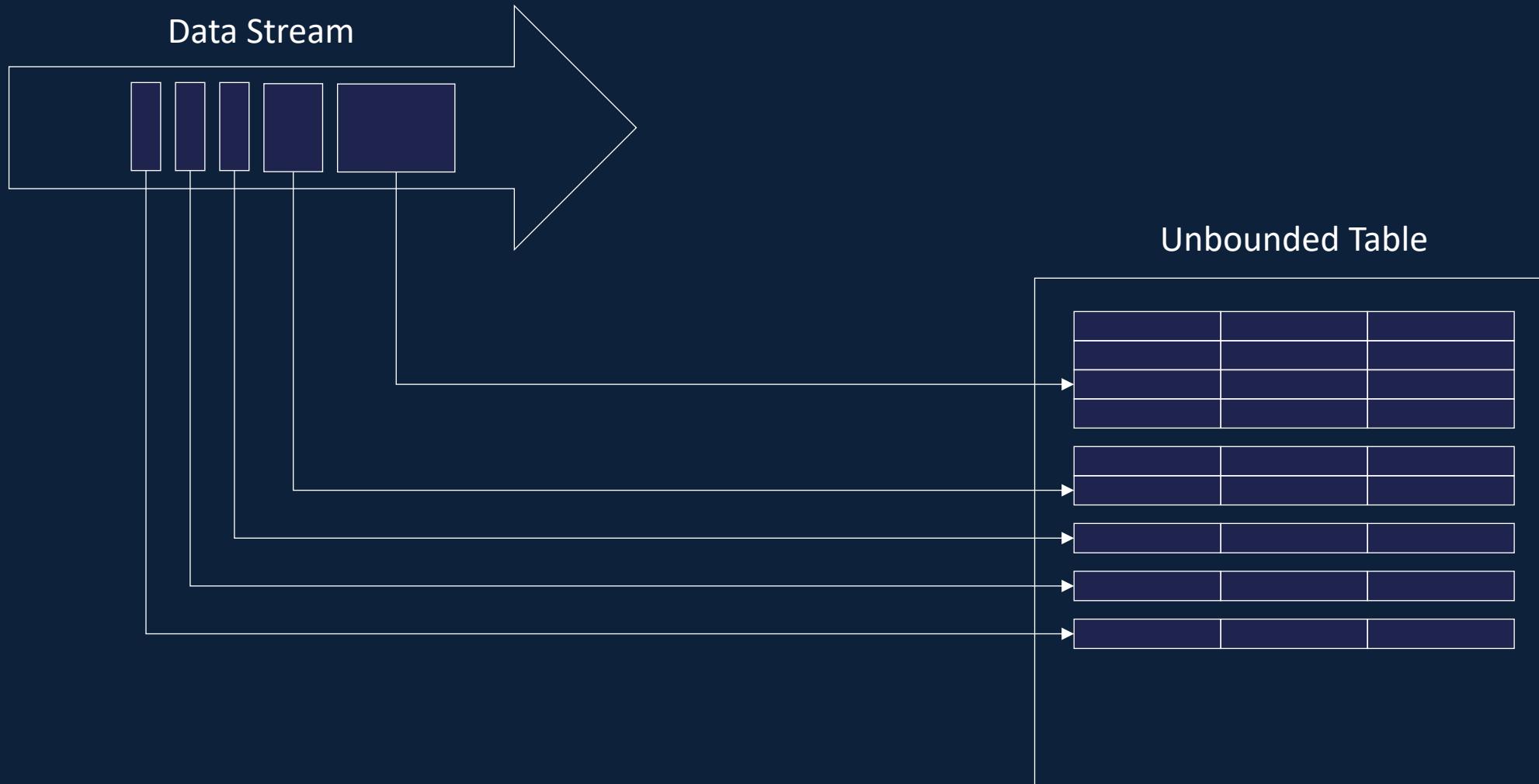
Exactly once guarantees

Built-in Fault tolerance

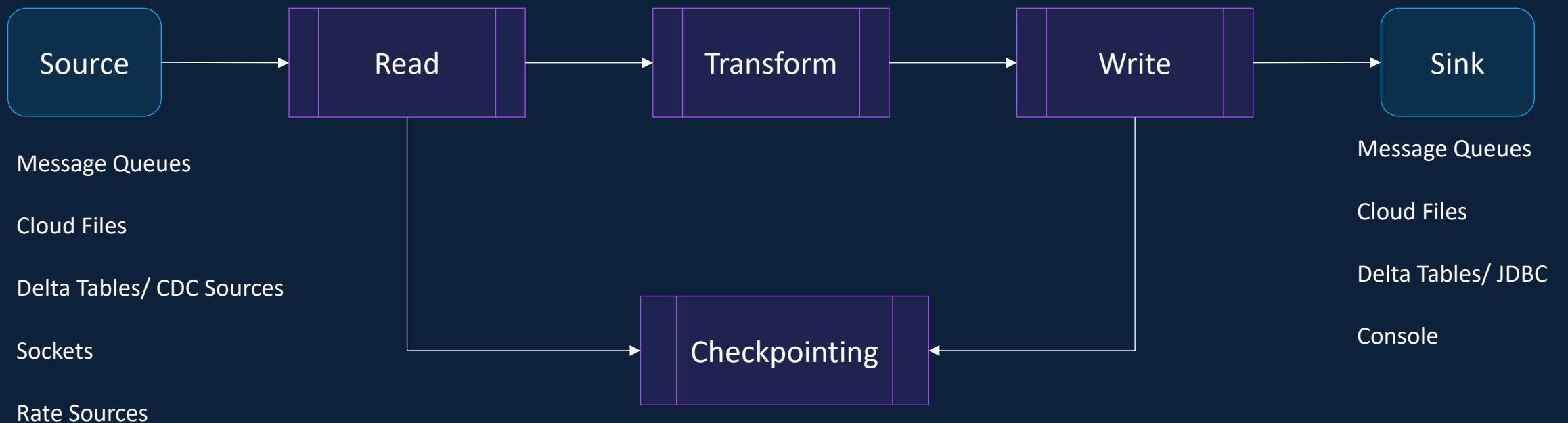
Scalable to large scale workloads

Supports various sources and sinks

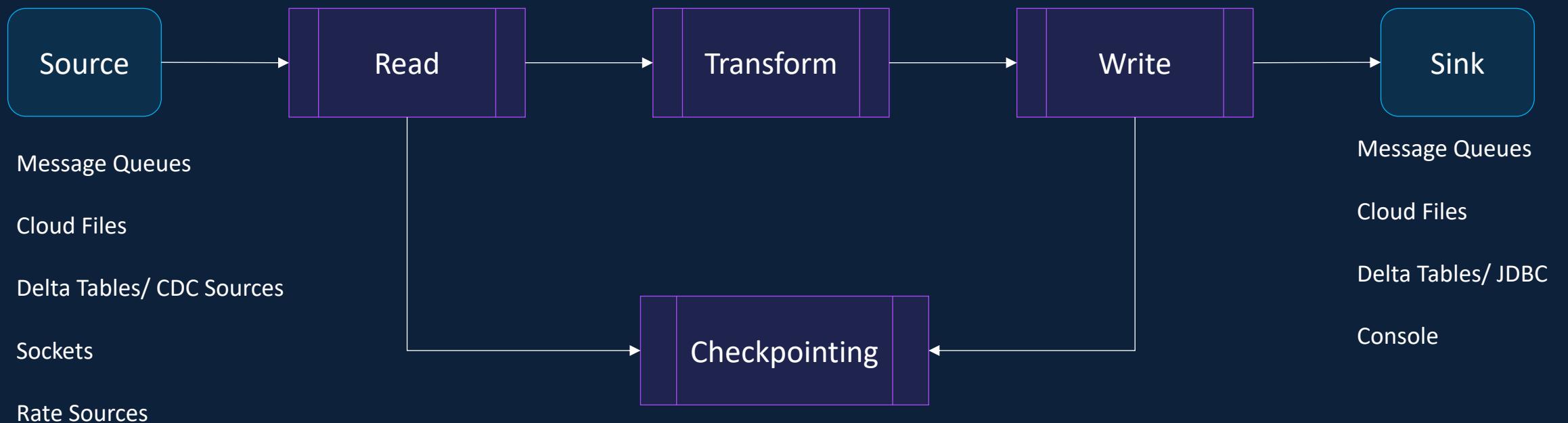
Spark Structured Streaming



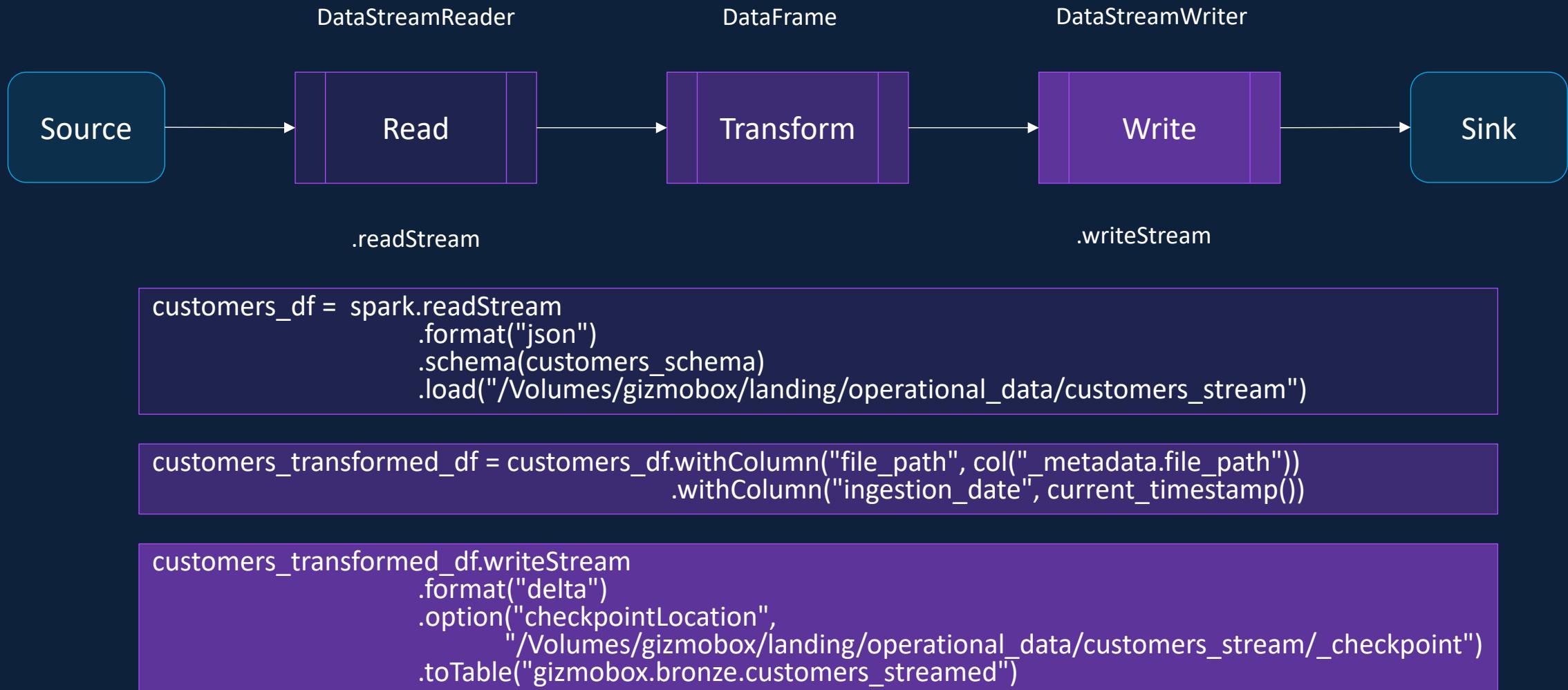
Spark Structured Streaming



Spark Structured Streaming



Spark Structured Streaming



Spark Structured Streaming

Trigger

Output Mode

Spark Structured Streaming

Processing Modes

Micro-Batch Processing (Default)

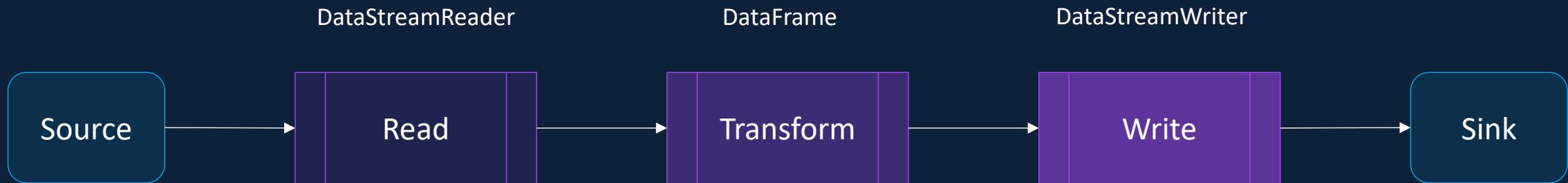
Continuous Processing (Experimental)

Spark Structured Streaming

Trigger

Trigger Type	Trigger Syntax	Description
Default (No Trigger)	No trigger specified	500 microseconds interval
Fixed Interval	.trigger(processingTime = "2 minutes")	Micro batch starts at the user specified interval
Triggered Once	.trigger(once=True)	Processes all data available as one micro-batch and stops [Deprecated]
Available Now	.trigger(availableNow=True)	Processes all data available as multiple micro-batch and stops
Continuous	.trigger(continuous="2 seconds")	Processes data continuously, but checkpoints at interval specified [Experimental]

Spark Structured Streaming



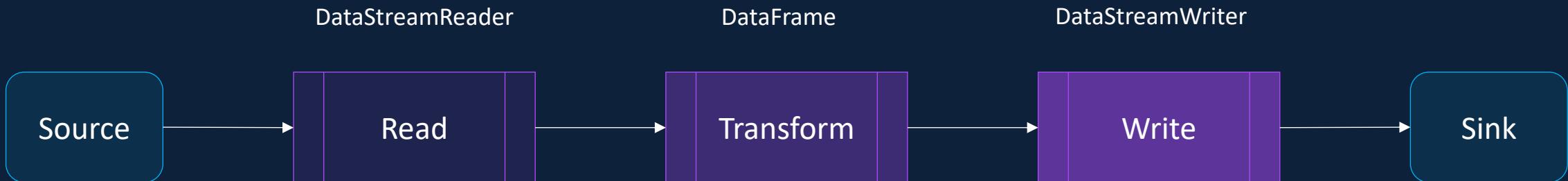
```
customers_transformed_df.writeStream  
    .format("delta")  
    .trigger(processingTime = "2 minutes")  
    .option("checkpointLocation",  
            "/Volumes/gizmobox/landing/operational_data/customers_stream/_checkpoint")  
    .toTable("gizmobox.bronze.customers_streamed")
```

Spark Structured Streaming

outputMode

Output Mode	Description
Append (Default)	Writes the new rows arrived since the last micro-batch
Complete	Writes the entire result to the sink every time
Update	Writes only the rows that have changed since the last micro-batch

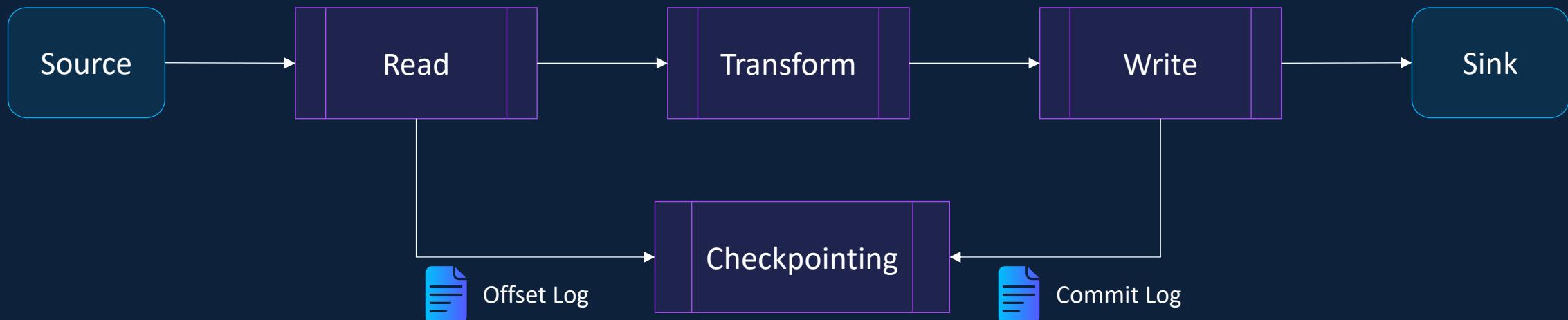
Spark Structured Streaming



```
customers_transformed_df.writeStream  
    .format("delta")  
    .outputMode("append")  
    .trigger(processingTime = "2 minutes")  
    .option("checkpointLocation",  
            "/Volumes/gizmobox/landing/operational_data/customers_stream/_checkpoint")  
    .toTable("gizmobox.bronze.customers_streamed")
```

Spark Structured Streaming

Checkpointing



Checkpointing is a fault-tolerance mechanism that allows a query to recover from failures and resume processing from where it left off without data loss or duplication.



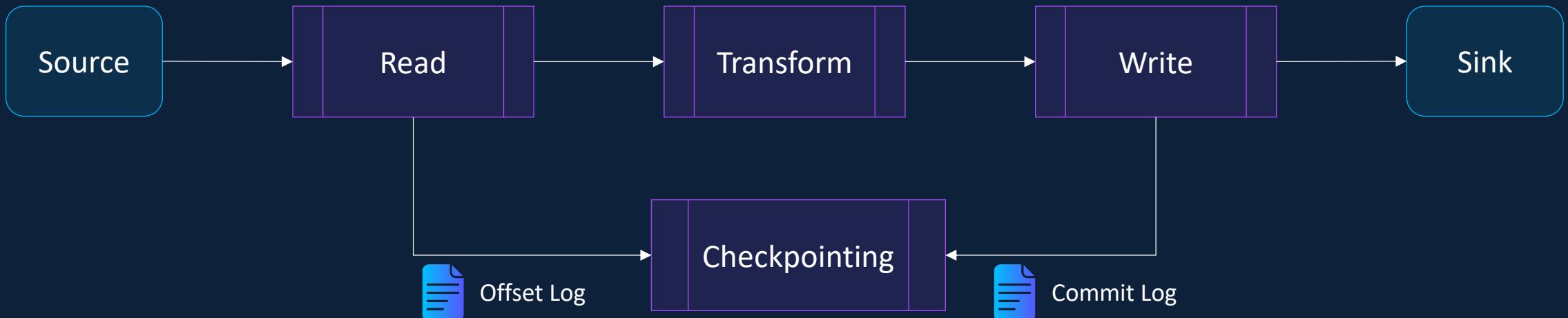
CheckPoint

Stores Metadata about streaming query, execution plan

Tracks processed offsets and committed results

Spark Structured Streaming

Checkpointing



Write-Ahead Logs (WAL) and Checkpointing helps provide Fault Tolerance

Idempotent Sinks enables exactly once guarantees

Auto Loader

Auto Loader

Auto Loader is a new structured streaming source designed for large-scale, efficient data ingestion

Auto Loader incrementally and efficiently processes new data files as they arrive in the cloud storage

Spark Structured Streaming



```
customers_df = spark.readStream  
    .format("json")  
    .schema(customers_schema)  
    .load("/Volumes/gizmobox/landing/operational_data/customers_stream")
```

Spark Structured Streaming

Traditional File Stream Source

Inefficient File Listing

Scalability Issues

Schema Evolution Problems

Spark Structured Streaming

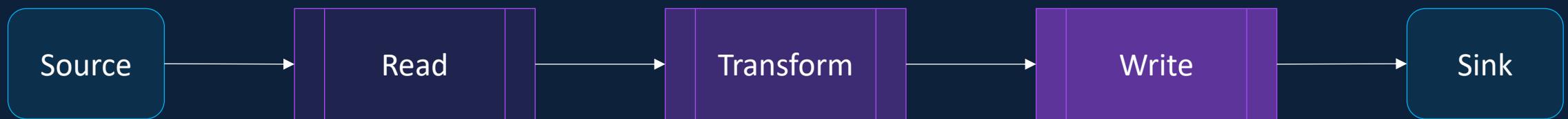
Auto Loader

Efficient File Detection using Cloud Services

Scalability Improvements

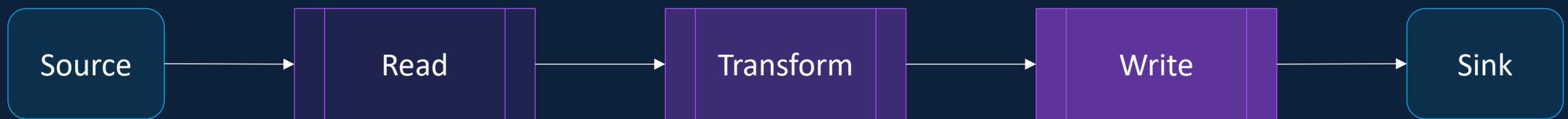
Schema Evolution & Resiliency

Auto Loader



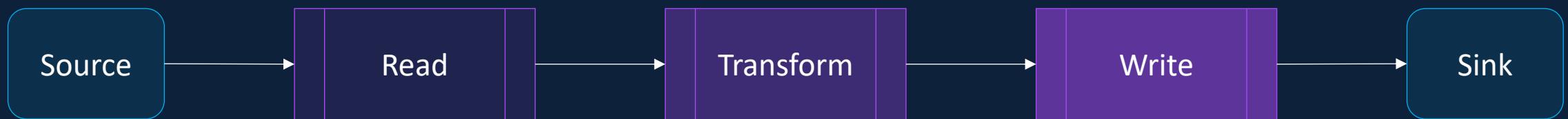
```
customers_df = spark.readStream  
    .format("json")  
    .schema(customers_schema)  
    .load("/Volumes/gizmobox/landing/operational_data/customers_stream")
```

Auto Loader



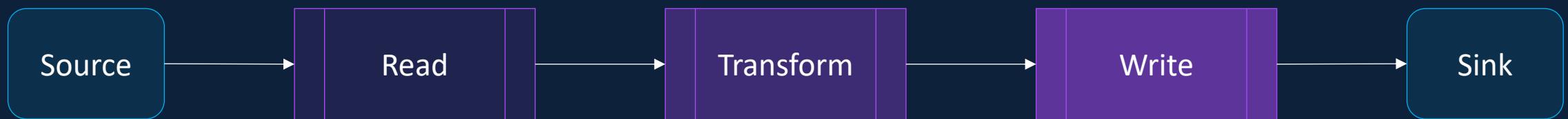
```
customers_df = spark.readStream  
    .format("cloudFiles")  
    .option("cloudFiles.format", "json")  
    .schema(customers_schema)  
    .load("/Volumes/gizmobox/landing/operational_data/customers_stream")
```

Auto Loader



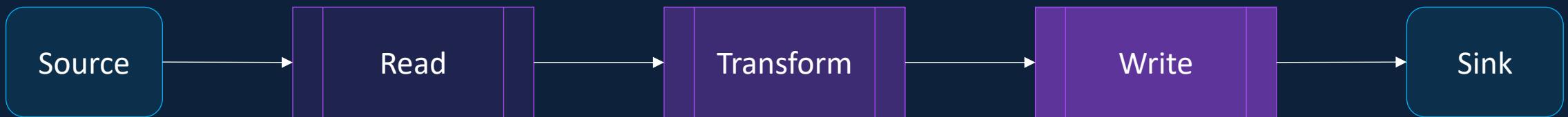
```
customers_df = spark.readStream  
    .format("cloudFiles")  
    .option("cloudFiles.format", "json")  
    .option("cloudFiles.useNotifications", "true")  
    .schema(customers_schema)  
    .load("/Volumes/gizmobox/landing/operational_data/customers_stream")
```

Auto Loader



```
customers_df = spark.readStream  
    .format("cloudFiles")  
    .option("cloudFiles.format", "json")  
    .option("cloudFiles.useNotifications", "true")  
    .option("cloudFiles.schemaLocation",  
           "/Volumes/gizmobox/landing/operational_data/customers_stream/_schema")  
    .option("cloudFiles.inferColumnTypes", "true")  
    .load("/Volumes/gizmobox/landing/operational_data/customers_stream")
```

Auto Loader



```
customers_df = spark.readStream  
    .format("cloudFiles")  
    .option("cloudFiles.format", "json")  
    .option("cloudFiles.useNotifications", "true")  
    .option("cloudFiles.schemaLocation",  
            "/Volumes/gizmobox/landing/operational_data/customers_stream/_schema")  
    .option("cloudFiles.inferColumnTypes", "true")  
    .option("cloudFiles.schemaEvolutionMode", "addNewColumns")  
    .load("/Volumes/gizmobox/landing/operational_data/customers_stream")
```



What is Delta Lake?

What is Delta Lake?



Delta Lake is the optimized storage layer that provides the foundation for tables in a Lakehouse on Databricks. Delta Lake is open source software that extends Parquet data files with a file-based transaction log for ACID transactions and scalable metadata handling.

It allows you to use a single copy of data for both batch and streaming operations and providing incremental processing at scale

- *Databricks*

What is Delta Lake?



Delta Lake is the **optimized storage layer** that provides the foundation for tables in a Lakehouse on Databricks. Delta Lake is open source software that extends Parquet data files with a file-based transaction log for **ACID transactions** and scalable metadata handling.

It allows you to use a single copy of data for both batch and **streaming operations** and providing incremental processing at scale

- *Databricks*

What is Delta Lake?

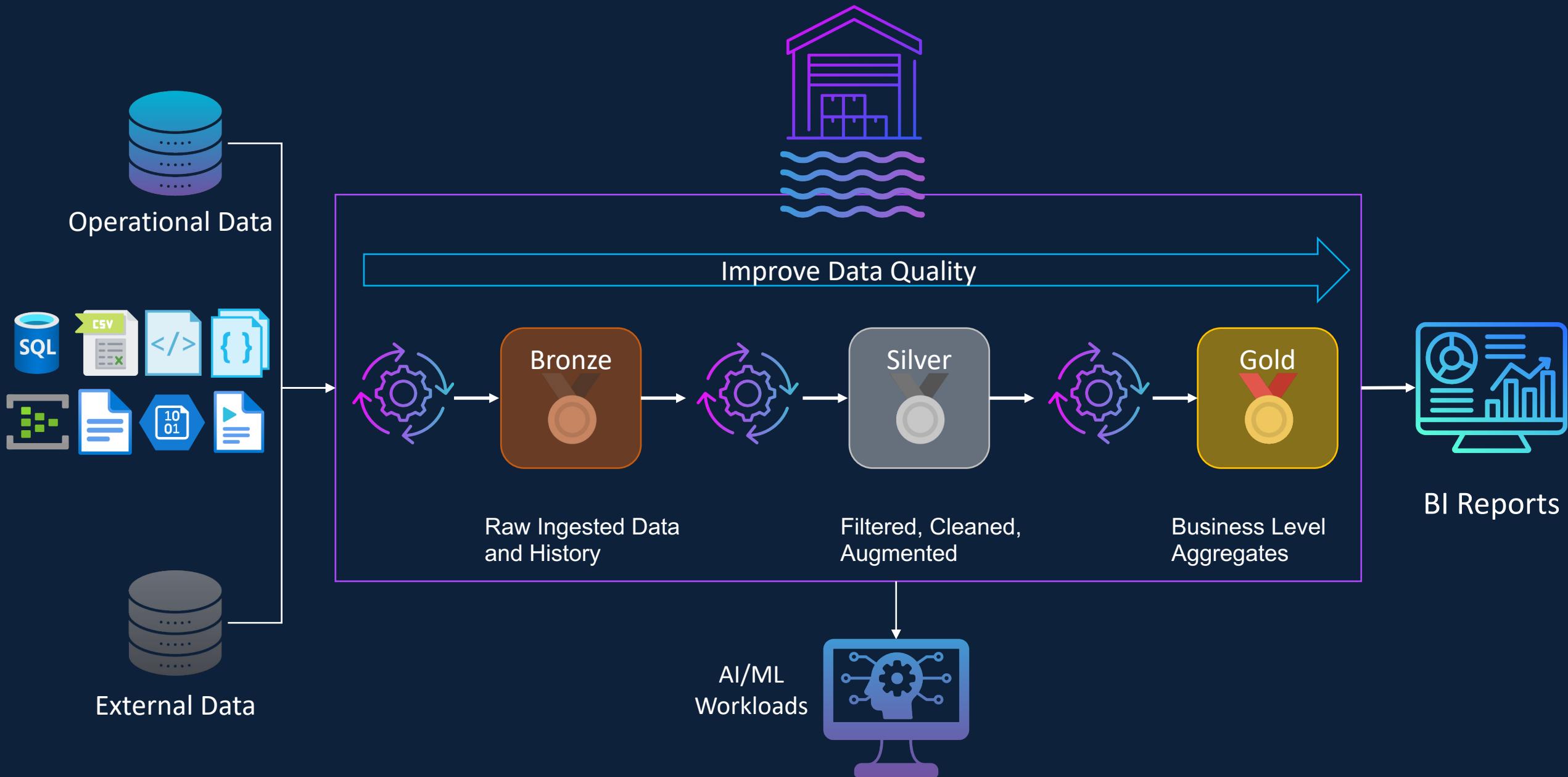


Delta Lake is the optimized storage layer that provides the foundation for tables in a Lakehouse on Databricks. Delta Lake is open source software that extends Parquet data files with a file-based transaction log for ACID transactions and scalable metadata handling.

It allows you to use a single copy of data for both batch and streaming operations and providing incremental processing at scale

- *Databricks*

What is Delta Lake?



Delta Lake – Key Characteristics



ACID Transactions

Atomicity

Consistency

Isolation

Durability

Delta Lake – Key Characteristics



ACID Transactions

Scalable Metadata

Time Travel

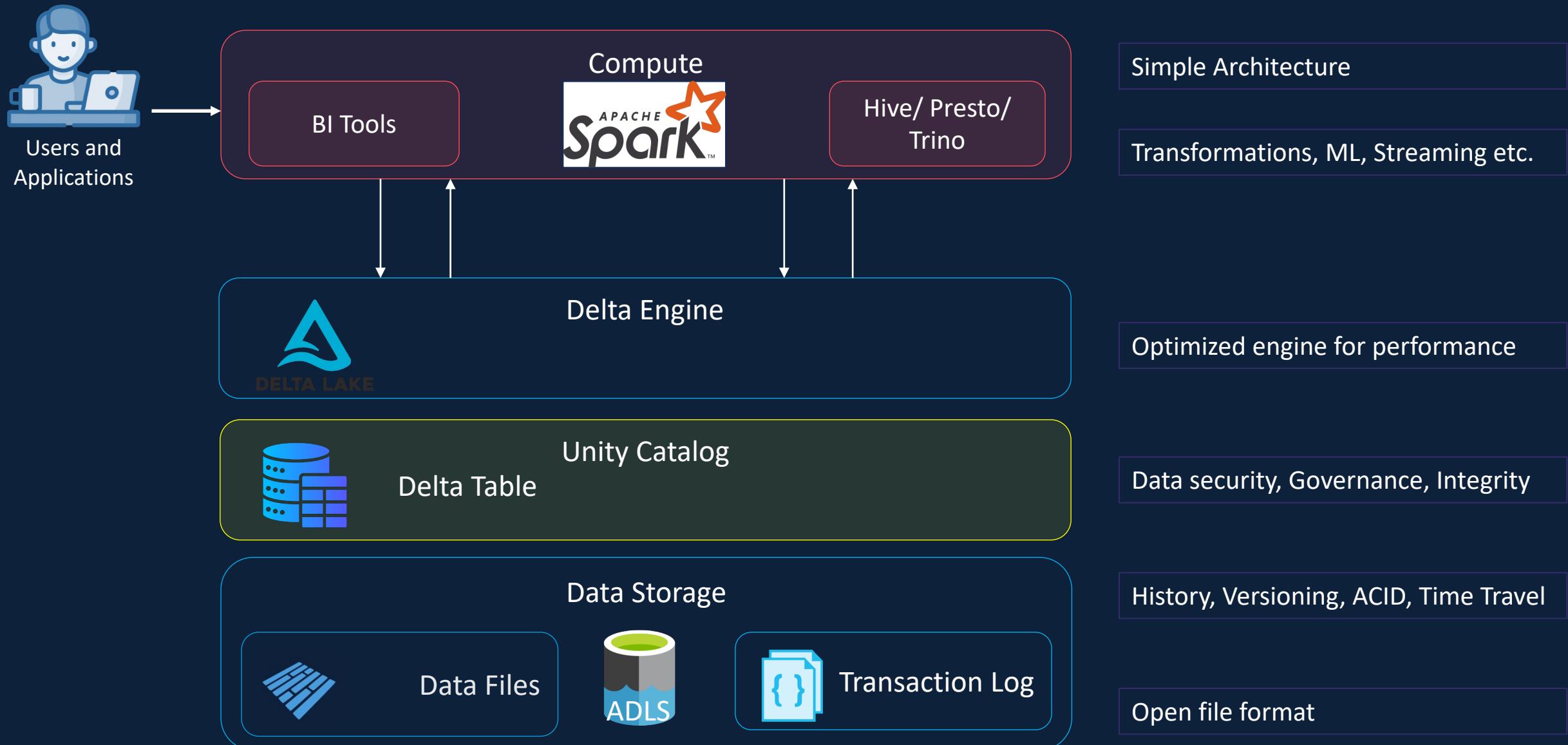
Simple Solution Architecture

Support for DML Operations

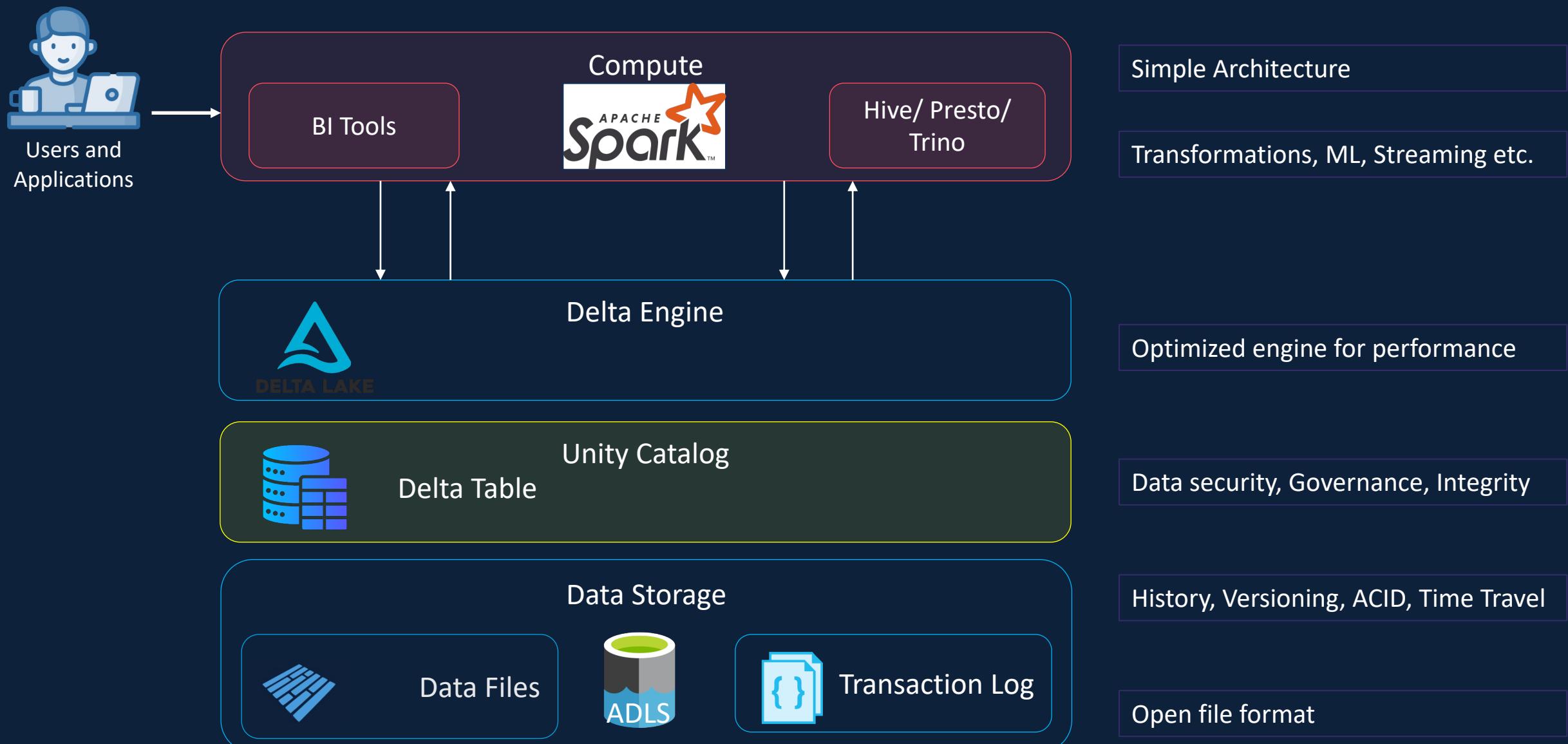
Better Performance

Open Source

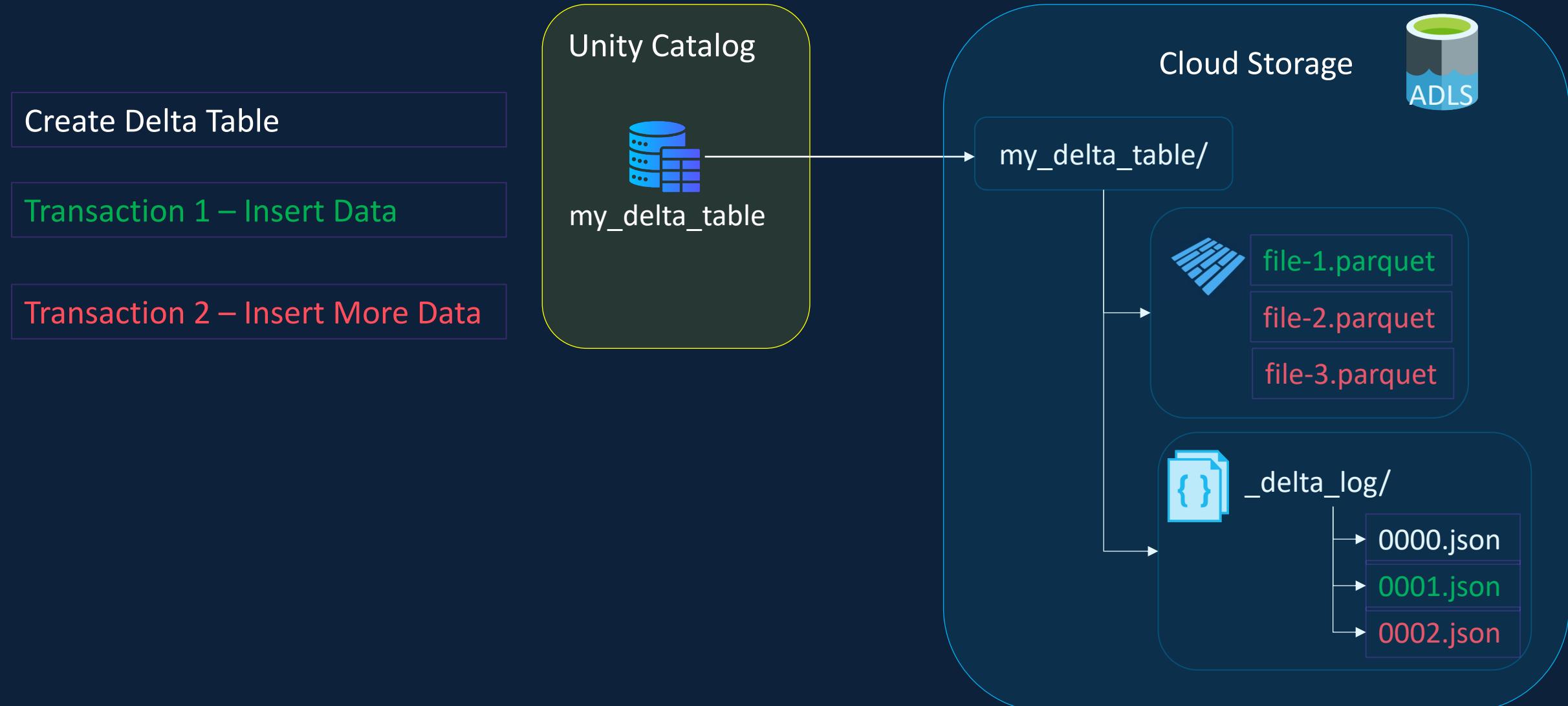
Delta Lake – Architecture



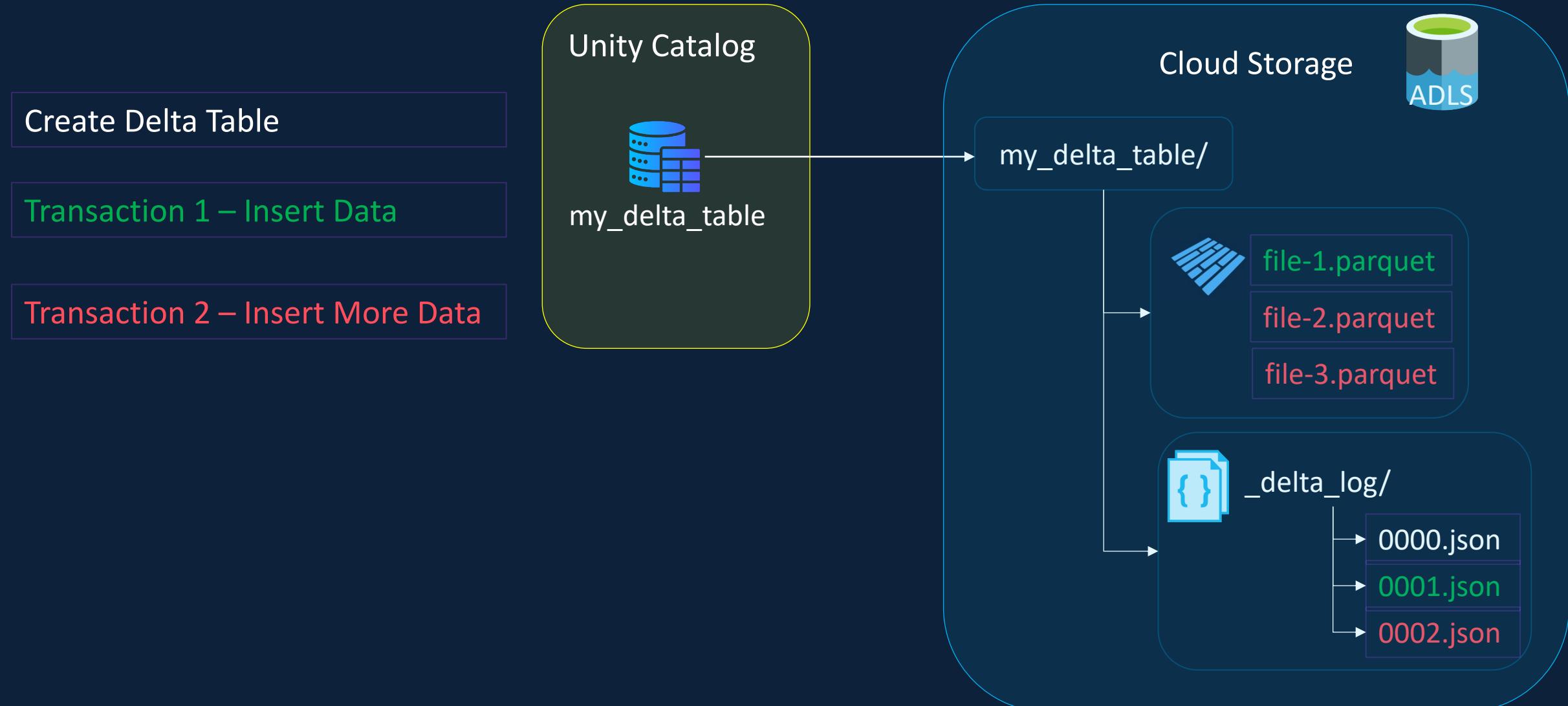
Delta Transaction Log



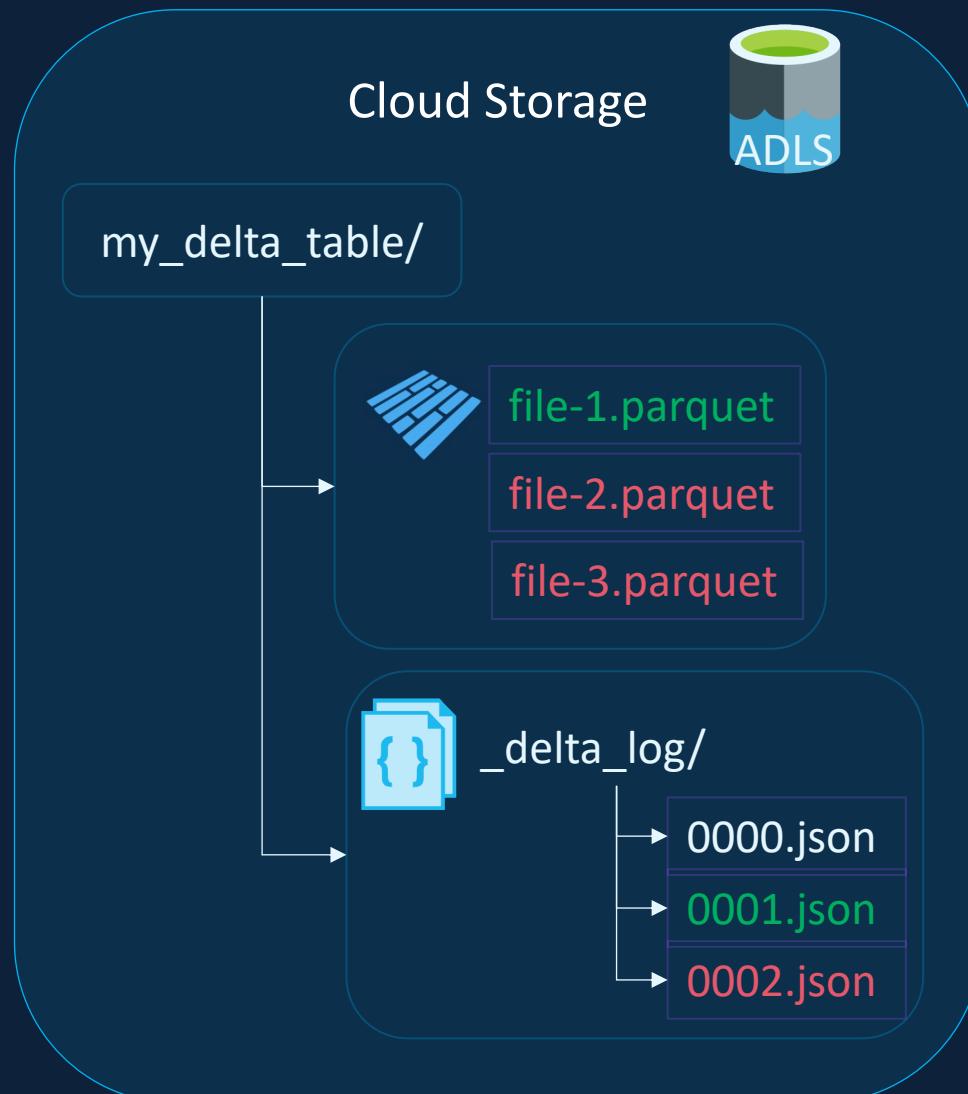
Delta Transaction Log



Delta Lake – ACID Transactions

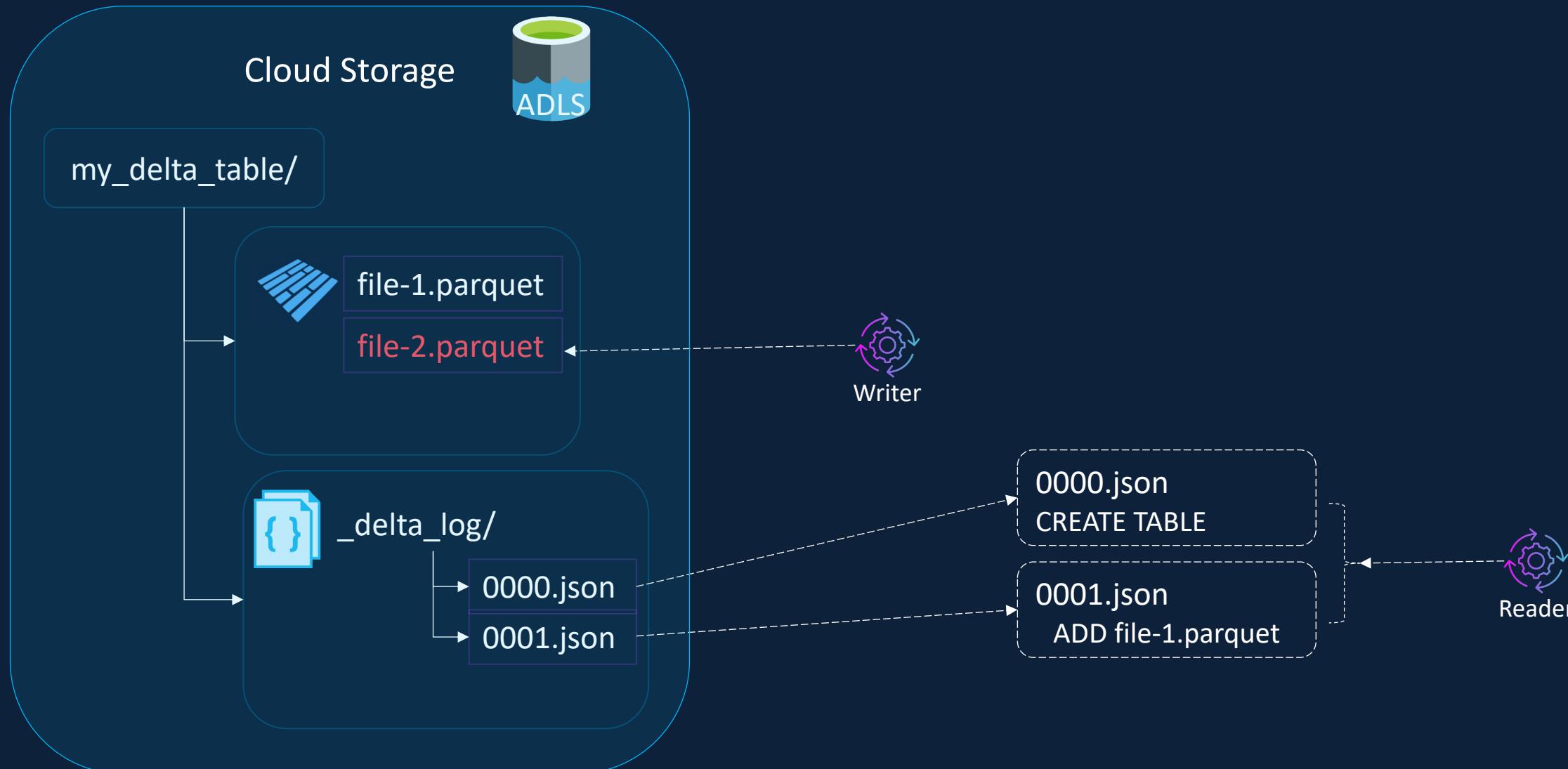


Delta Lake – ACID Transactions

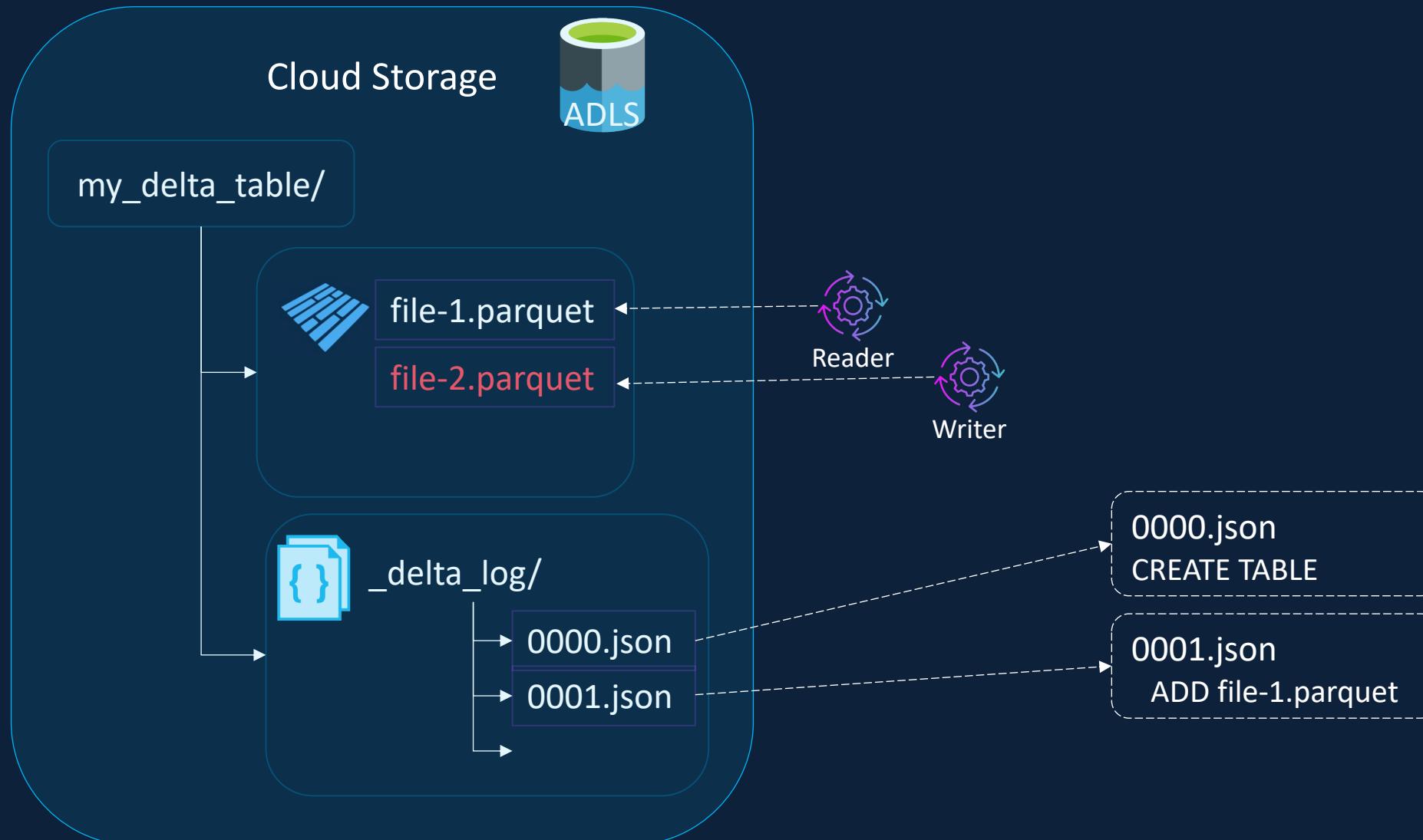


1. Transaction Logs are written at the end of the transaction
2. Readers will always read the transaction logs first to identify the list of data files to read

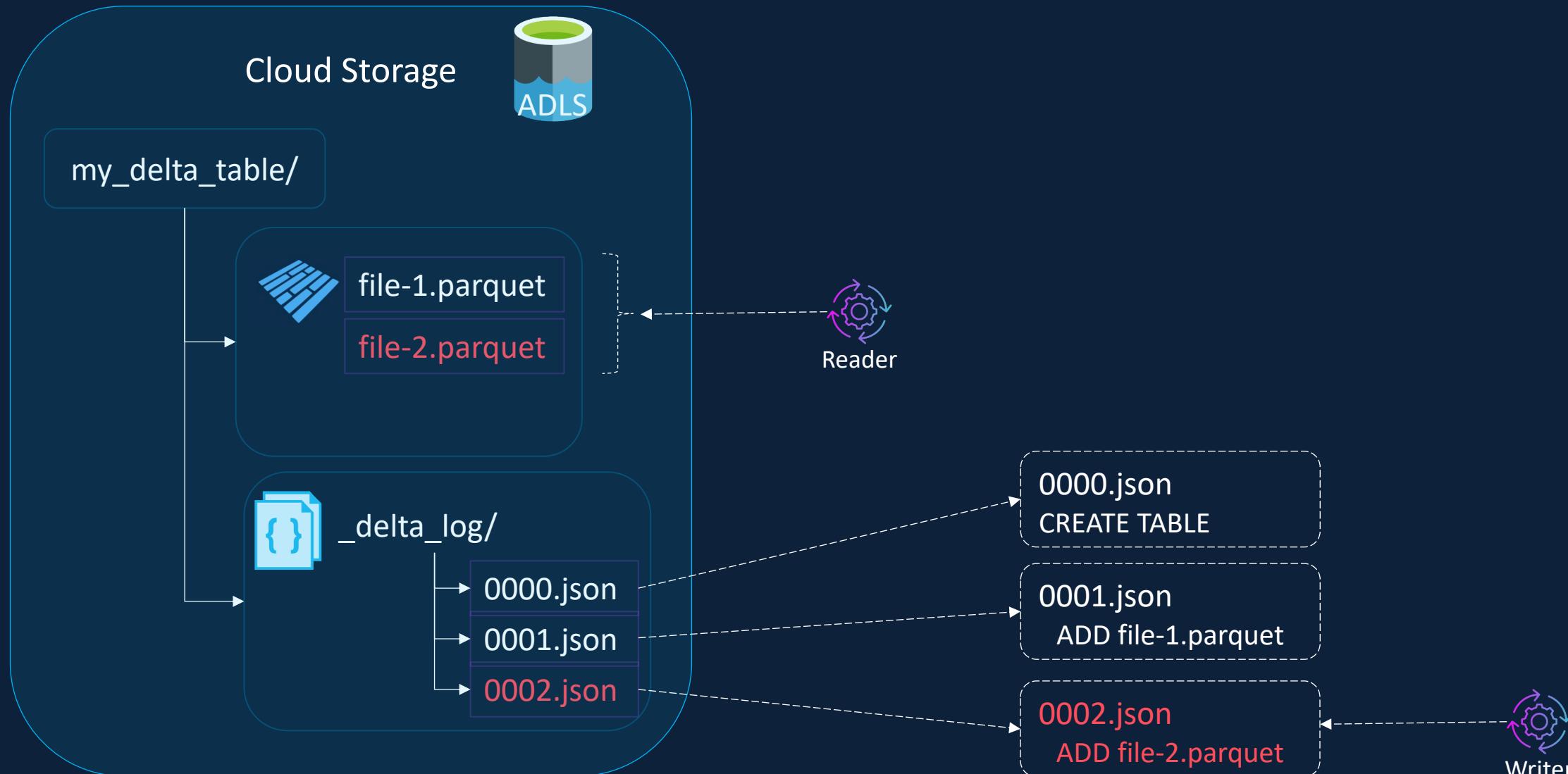
Delta Lake – ACID Transactions



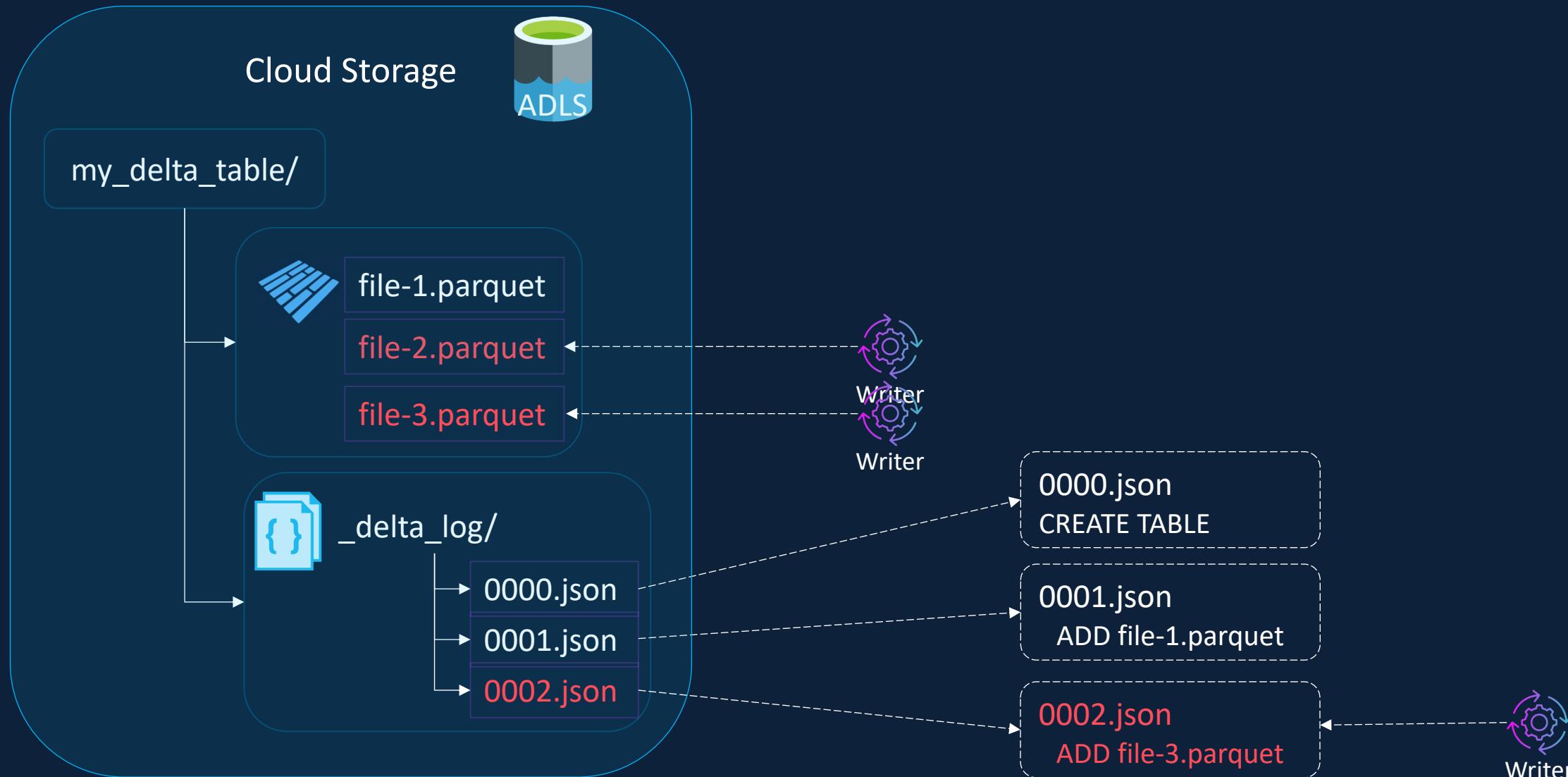
Delta Lake – ACID Transactions



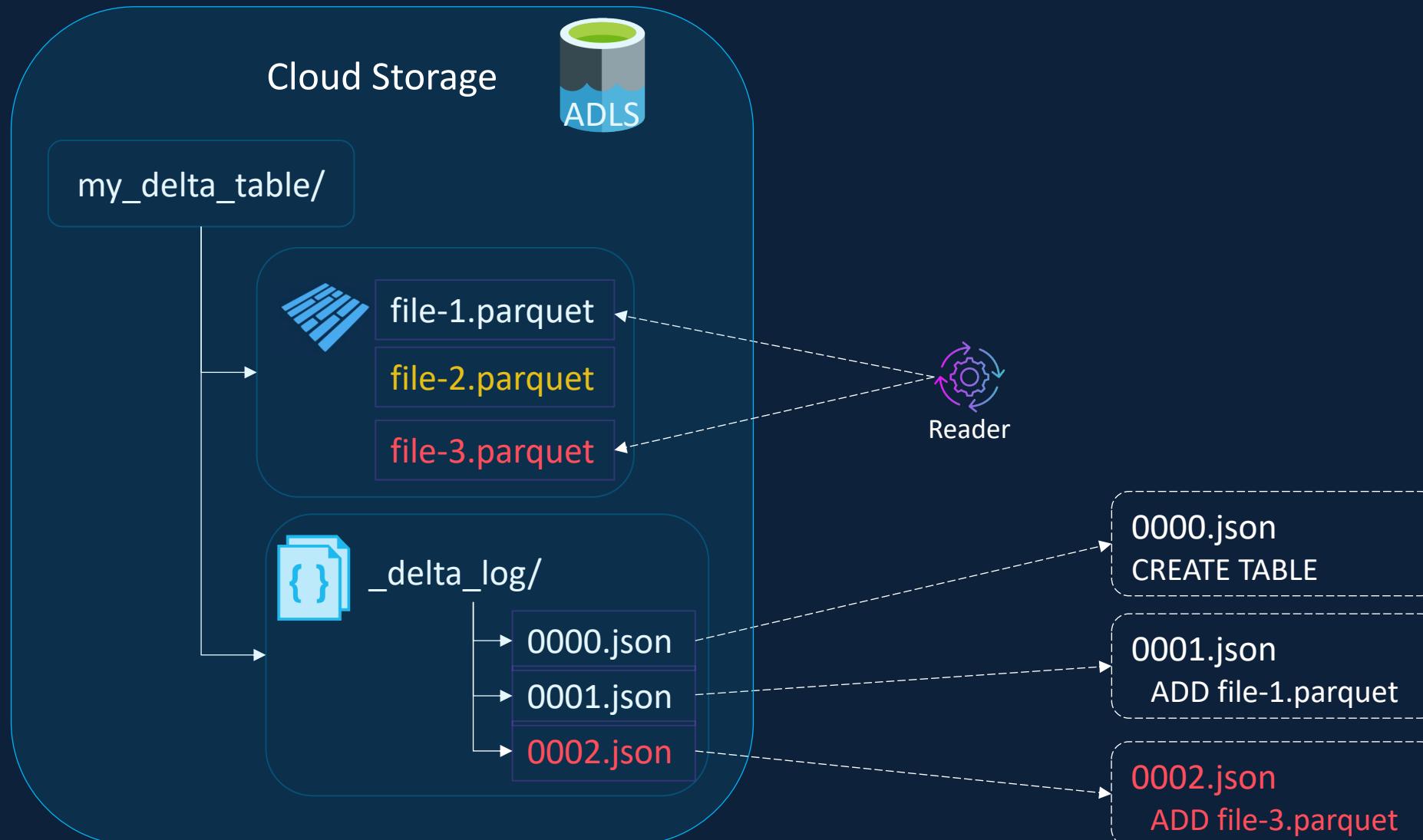
Delta Lake – ACID Transactions



Delta Lake – ACID Transactions



Delta Lake – ACID Transactions



Delta Lake – CREATE TABLE

```
{ { [CREATE OR] REPLACE TABLE |  
    CREATE [EXTERNAL] TABLE [ IF NOT EXISTS ] }  
table_name  
[ table_specification ]  
[ USING data_source ]  
[ table_clauses ]  
[ AS query ] }
```

```
table_specification  
( { column_identifier column_type [ column_properties ] } [, ...]  
  [ , table_constraint ] [...] )
```

```
column_properties  
{ NOT NULL |  
  COLLATE collation_name |  
  GENERATED ALWAYS AS ( expr ) |  
  GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY [ ( [ START WITH start ] [  
INCREMENT BY step ] ) ] |  
  DEFAULT default_expression |  
  COMMENT column_comment |  
  column_constraint |  
  MASK clause } [ ... ]
```

```
table_clauses  
{ OPTIONS clause |  
  PARTITIONED BY clause |  
  CLUSTER BY clause |  
  clustered_by_clause |  
  LOCATION path [ WITH ( CREDENTIAL credential_name ) ] |  
  COMMENT table_comment |  
  TBLPROPERTIES clause |  
  WITH { ROW FILTER clause } } [...]
```

Delta Lake – CREATE TABLE

```
{ { [CREATE OR] REPLACE TABLE |  
    CREATE [EXTERNAL] TABLE [ IF NOT EXISTS ] }  
table_name  
[ table_specification ]  
[ USING data_source ]  
[ table_clauses ]  
[ AS query ] }
```

```
table_specification  
( { column_identifier column_type [ column_properties ] } [, ...]  
  [ , table_constraint ] [...] )
```

```
column_properties  
{ NOT NULL |  
  COLLATE collation_name |  
  GENERATED ALWAYS AS ( expr ) |  
  GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY [ ( [ START  
WITH start ] [ INCREMENT BY step ] ) ] |  
  DEFAULT default_expression |  
  COMMENT column_comment |  
  column_constraint |  
  MASK clause } [ ... ]
```

```
table_clauses  
{ OPTIONS clause |  
  PARTITIONED BY clause |  
  CLUSTER BY clause |  
  clustered_by_clause |  
  LOCATION path [ WITH ( CREDENTIAL credential_name ) ] |  
  COMMENT table_comment |  
  TBLPROPERTIES clause |  
  WITH { ROW FILTER clause } } [ ... ]
```

Delta Lake – CREATE TABLE

```
{ { [CREATE OR] REPLACE TABLE |  
    CREATE [EXTERNAL] TABLE [ IF NOT EXISTS ] }  
table_name  
[ table_specification ]  
[ USING data_source ]  
[ table_clauses ]  
[ AS query ] }
```

table_specification
({ column_identifier column_type [column_properties] } [, ...]
[, table_constraint] [...])

column_properties
{ NOT NULL |
COLLATE collation_name |
GENERATED ALWAYS AS (expr) |
GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY [([START
WITH start] [INCREMENT BY step])] |
DEFAULT default_expression |
COMMENT column_comment |
column_constraint |
MASK clause } [...]

table_clauses
{ OPTIONS clause |
PARTITIONED BY clause |
CLUSTER BY clause |
clustered_by_clause |
LOCATION path [WITH (CREDENTIAL credential_name)] |
COMMENT table_comment |
TBLPROPERTIES clause |
WITH { ROW FILTER clause } } [...]

Delta Lake – CREATE TABLE

```
{ { [CREATE OR] REPLACE TABLE |  
    CREATE [EXTERNAL] TABLE [ IF NOT EXISTS ] }  
table_name  
[ table_specification ]  
[ USING data_source ]  
[ table_clauses ]  
[ AS query ] }
```

```
table_specification  
( { column_identifier column_type [ column_properties ] } [, ...]  
  [ , table_constraint ] [...] )
```

```
column_properties  
{ NOT NULL |  
  COLLATE collation_name |  
  GENERATED ALWAYS AS ( expr ) |  
  GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY [ ( [ START  
WITH start ] [ INCREMENT BY step ] ) ] |  
  DEFAULT default_expression |  
  COMMENT column_comment |  
  column_constraint |  
  MASK clause } [ ... ]
```

```
table_clauses  
{ OPTIONS clause |  
  PARTITIONED BY clause |  
  CLUSTER BY clause |  
  clustered_by_clause |  
  LOCATION path [ WITH ( CREDENTIAL credential_name ) ] |  
  COMMENT table_comment |  
  TBLPROPERTIES clause |  
  WITH { ROW FILTER clause } } [ ... ]
```

Delta Lake – CREATE TABLE

```
{ { [CREATE OR] REPLACE TABLE |  
    CREATE [EXTERNAL] TABLE [ IF NOT EXISTS ] }  
table_name  
[ table_specification ]  
[ USING data_source ]  
[ table_clauses ]  
[ AS query ] }
```

```
table_specification  
( { column_identifier column_type [ column_properties ] } [, ...]  
  [ , table_constraint ] [...] )
```

```
column_properties  
{ NOT NULL |  
  COLLATE collation_name |  
  GENERATED ALWAYS AS ( expr ) |  
  GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY [ ( [ START  
WITH start ] [ INCREMENT BY step ] ) ] |  
  DEFAULT default_expression |  
  COMMENT column_comment |  
  column_constraint |  
  MASK clause } [ ... ]
```

```
table_clauses  
{ OPTIONS clause |  
  PARTITIONED BY clause |  
  CLUSTER BY clause |  
  clustered_by_clause |  
  LOCATION path [ WITH ( CREDENTIAL credential_name ) ] |  
  COMMENT table_comment |  
  TBLPROPERTIES clause |  
  WITH { ROW FILTER clause } } [ ... ]
```

Delta Lake – CREATE TABLE

```
{ { [CREATE OR] REPLACE TABLE |  
    CREATE [EXTERNAL] TABLE [ IF NOT EXISTS ] }  
table_name  
[ table_specification ]  
[ USING data_source ]  
[ table_clauses ]  
[ AS query ] }
```

```
table_specification  
( { column_identifier column_type [ column_properties ] } [, ...]  
  [ , table_constraint ] [...] )
```

```
column_properties  
{ NOT NULL |  
  COLLATE collation_name |  
  GENERATED ALWAYS AS ( expr ) |  
  GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY [ ( [ START  
WITH start ] [ INCREMENT BY step ] ) ] |  
  DEFAULT default_expression |  
  COMMENT column_comment |  
  column_constraint |  
  MASK clause } [ ... ]
```

```
table_clauses  
{ OPTIONS clause |  
  PARTITIONED BY clause |  
  CLUSTER BY clause |  
  clustered_by_clause |  
  LOCATION path [ WITH ( CREDENTIAL credential_name ) ] |  
  COMMENT table_comment |  
  TBLPROPERTIES clause |  
  WITH { ROW FILTER clause } } [ ... ]
```

Delta Lake – Compaction

Process of combining many smaller files into few larger files to improve performance and optimize storage.



Small File Problem

Faster Reads

Efficient Storage

Delta Lake – Compaction



File Compaction (OPTIMIZE Command)

Z-Order Compaction (ZORDER BY Clause)

Liquid Clustering (Preview Feature)

Delta Lake – Vacuum

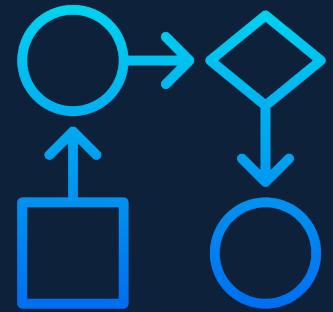
Used to remove old, unused files from the Delta Lake to free up storage. Permanently deletes files that are no longer referenced in the transaction log and older than retention threshold (default 7 days)



Reduces cloud storage costs

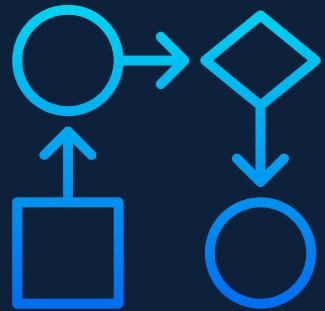
Improves query performance

Helps comply with privacy regulations



Databricks Workflows / Jobs

Databricks Workflows / Jobs

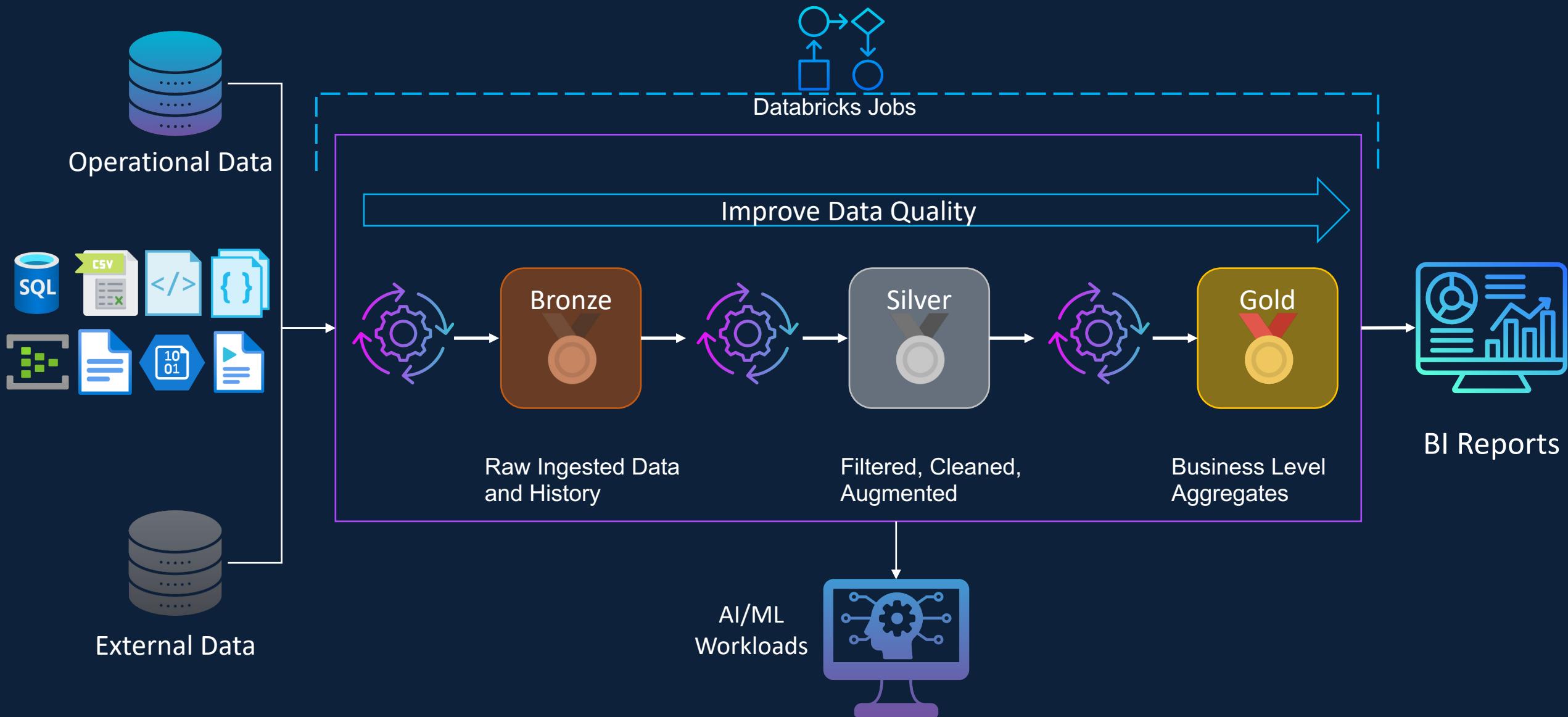


Databricks Workflows is a managed orchestration service, fully integrated with the Databricks Data Intelligence Platform.

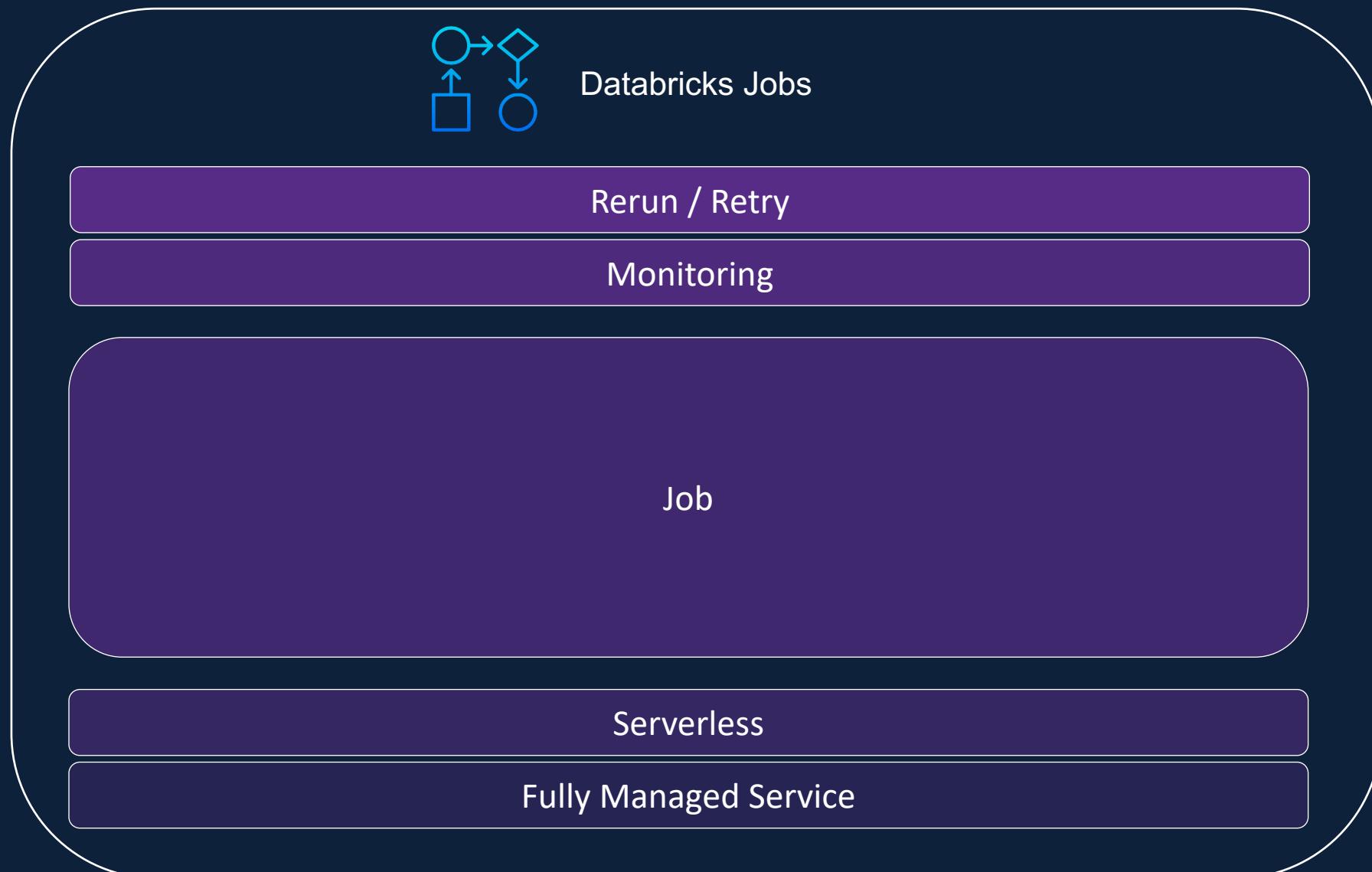
Workflows lets you easily define, manage and monitor multitask workflows for ETL, analytics and machine learning pipelines.

- *Databricks*

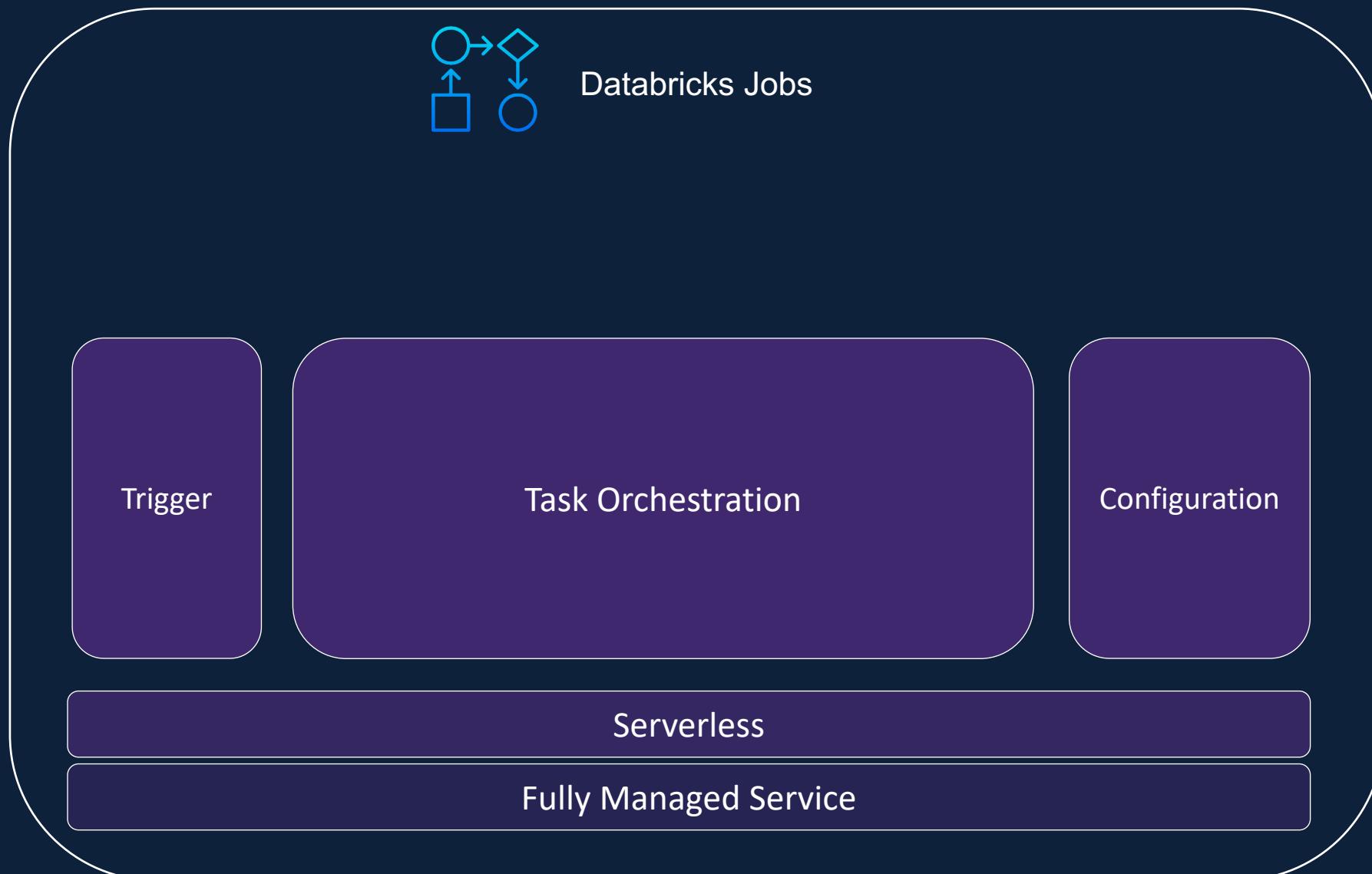
Databricks Workflows / Jobs



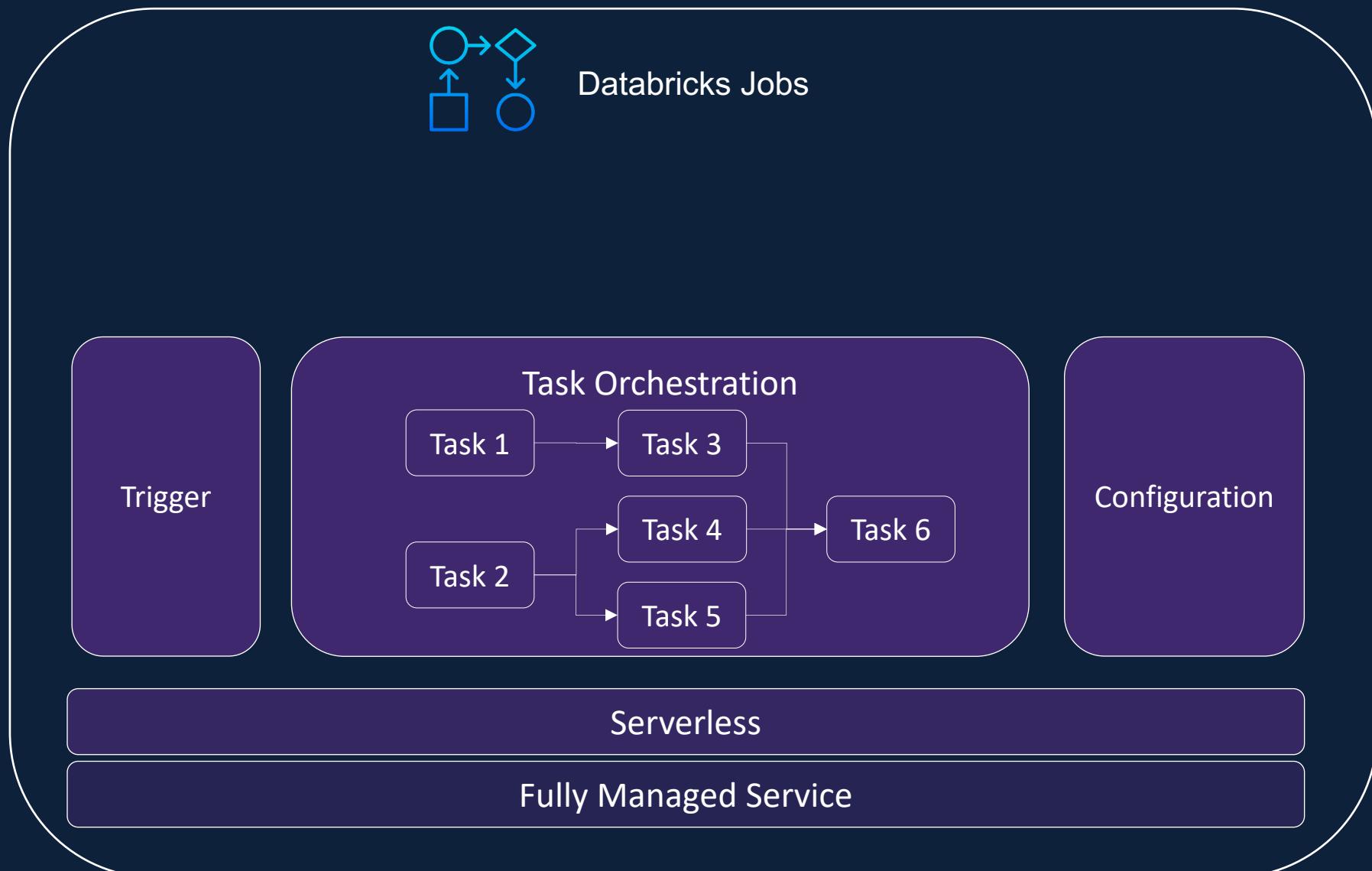
Databricks Workflows / Jobs



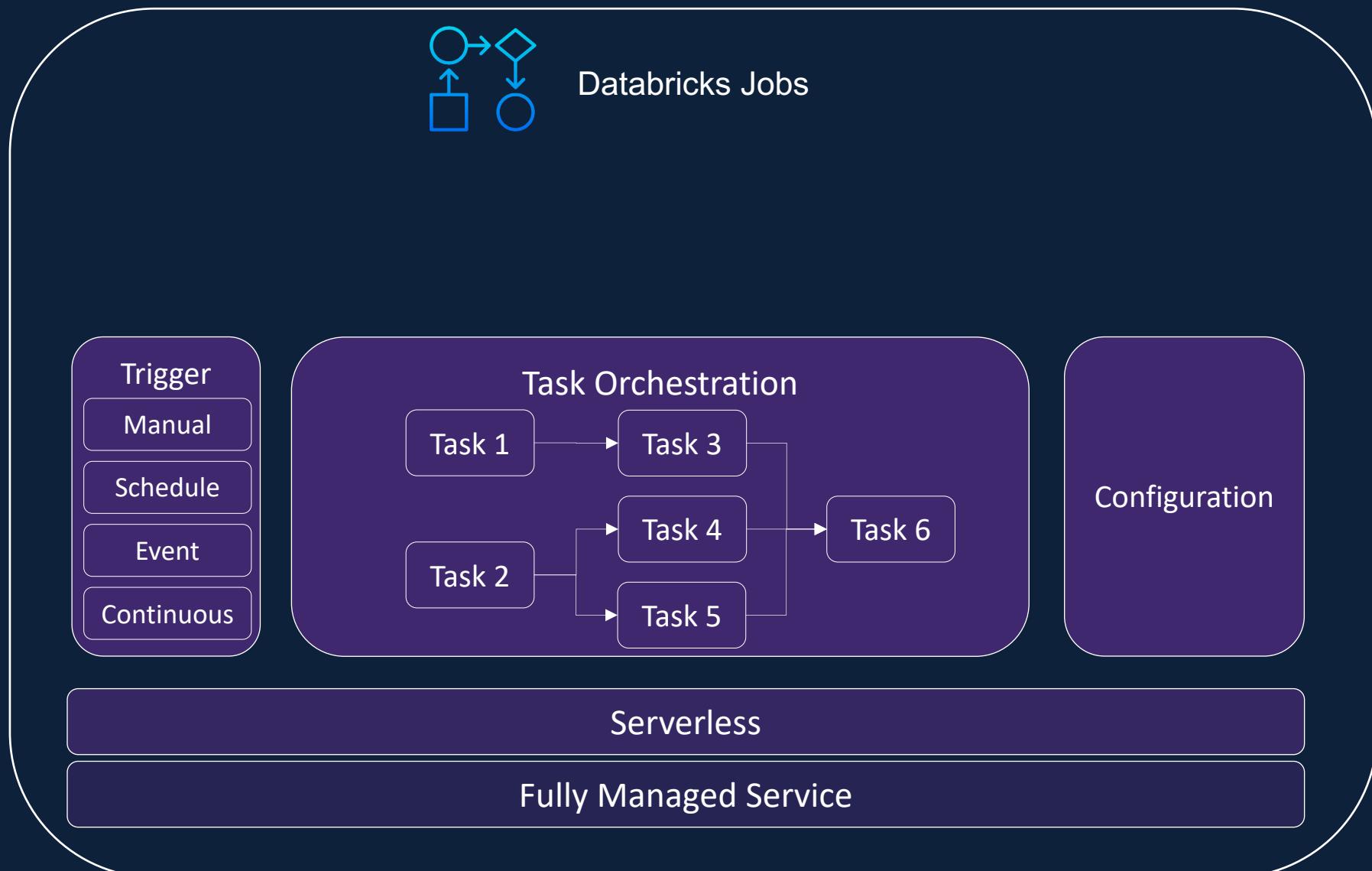
Databricks Workflows / Jobs



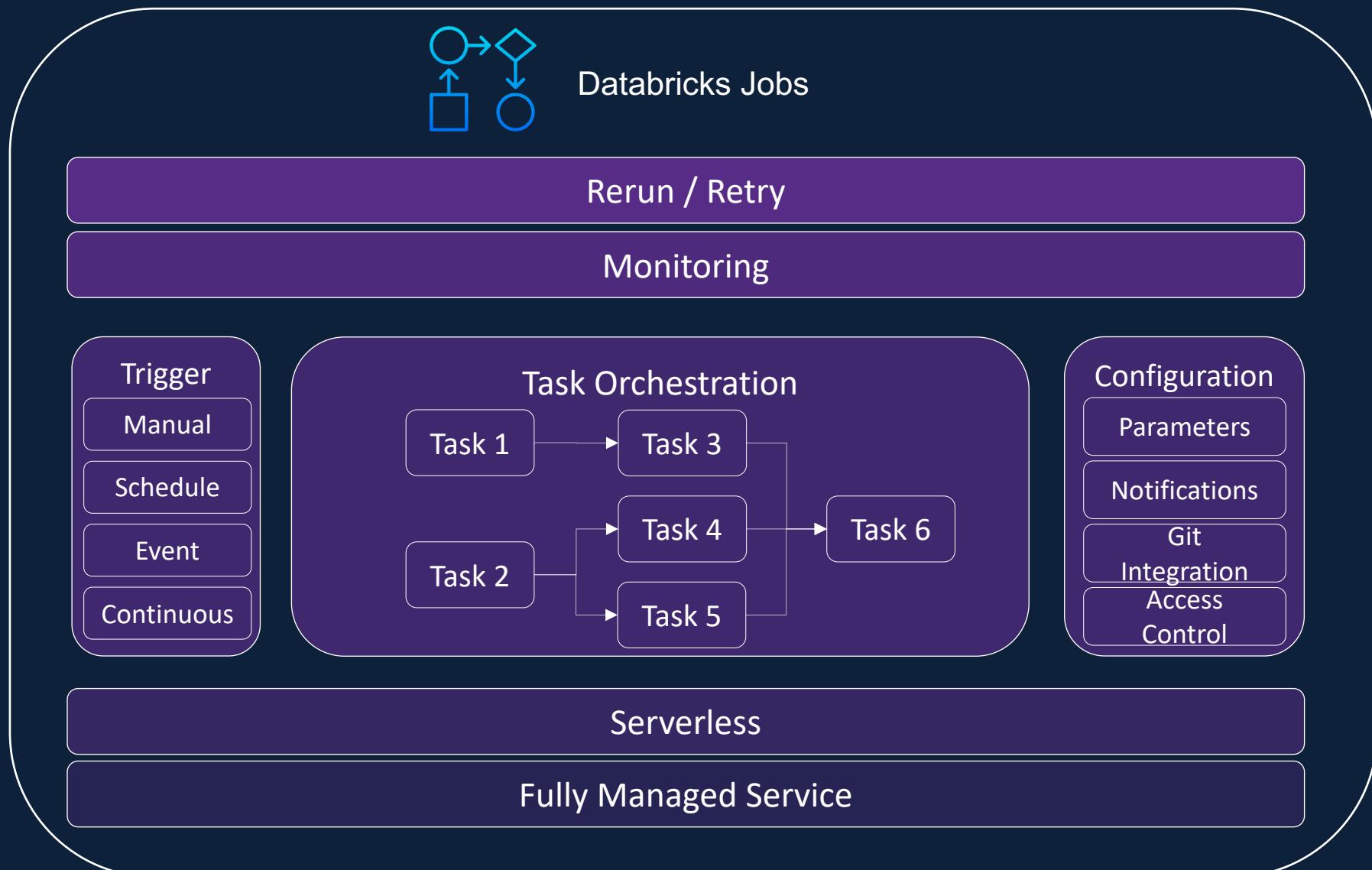
Databricks Workflows / Jobs



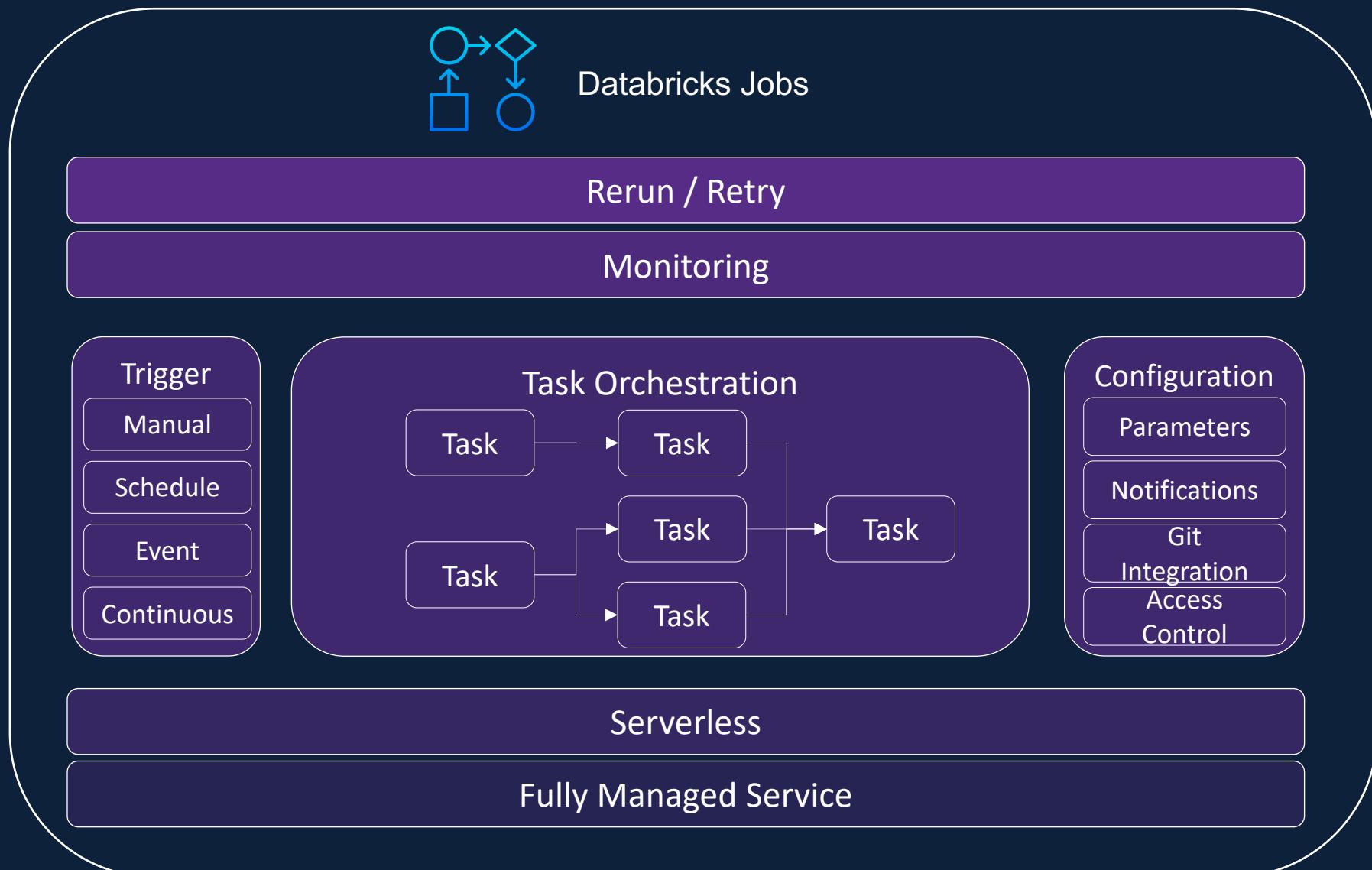
Databricks Workflows / Jobs



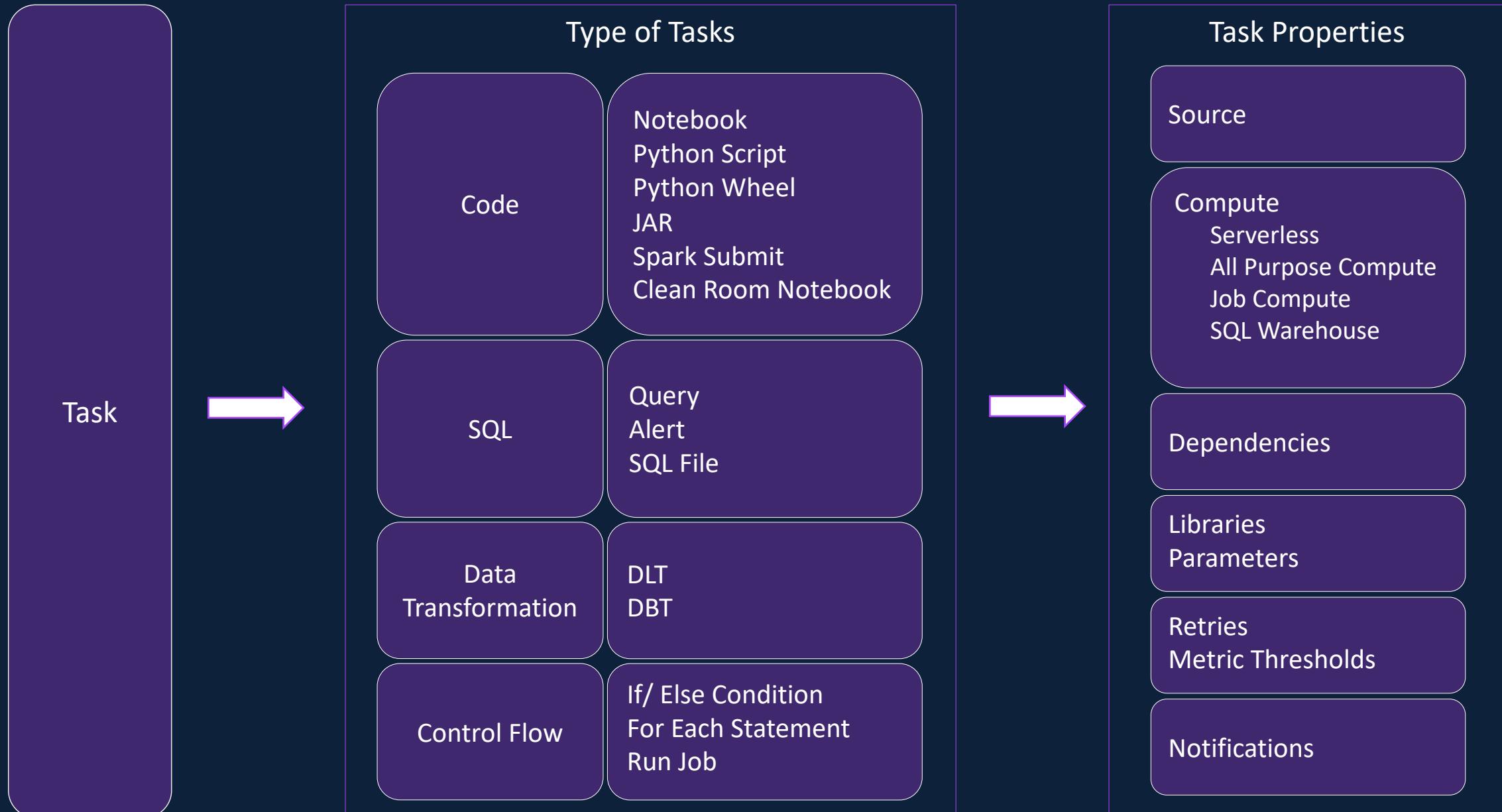
Databricks Workflows / Jobs



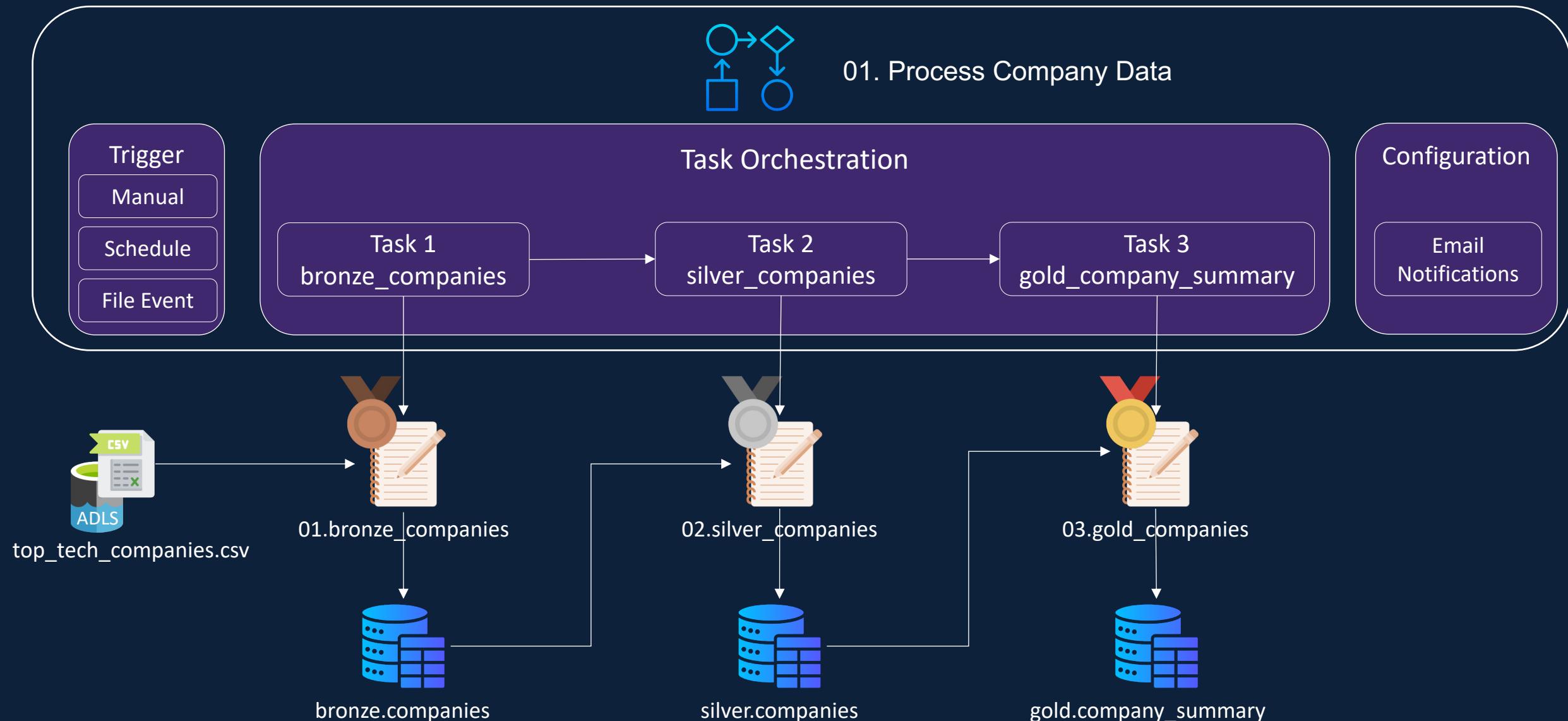
Databricks Workflows / Jobs



Databricks Workflows / Jobs



Databricks Workflows / Jobs





Delta Live Tables (DLT)



Delta Live Tables (DLT)

Delta Live Tables is a declarative ETL framework for building reliable, maintainable, and testable data processing pipelines.

You define the transformations to perform on your data and Delta Live Tables manages task orchestration, cluster management, monitoring, data quality, and error handling.

It handles both streaming and batch workloads with minimal manual intervention.

- *Databricks*



Delta Live Tables (DLT)

Delta Live Tables is a **declarative ETL framework** for building reliable, maintainable, and testable data processing pipelines.

You define the transformations to perform on your data and Delta Live Tables manages task orchestration, cluster management, monitoring, data quality, and error handling.

It handles both streaming and batch workloads with minimal manual intervention.

- *Databricks*



Delta Live Tables (DLT)

Delta Live Tables is a **declarative ETL framework** for building reliable, maintainable, and testable data processing pipelines.

You define the transformations to perform on your data and Delta Live Tables manages **task orchestration, cluster management, monitoring, data quality, and error handling**.

It handles both streaming and batch workloads with minimal manual intervention.

- Databricks



Delta Live Tables (DLT)

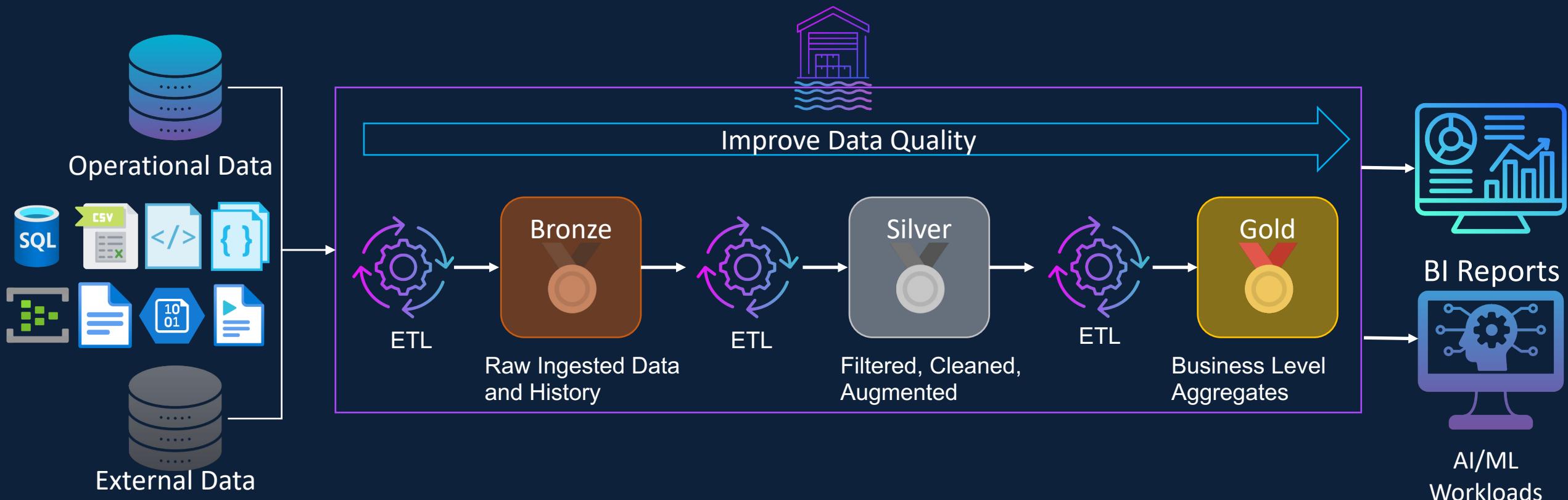
Delta Live Tables is a **declarative ETL framework** for building reliable, maintainable, and testable data processing pipelines.

You define the transformations to perform on your data and Delta Live Tables manages **task orchestration, cluster management, monitoring, data quality, and error handling**.

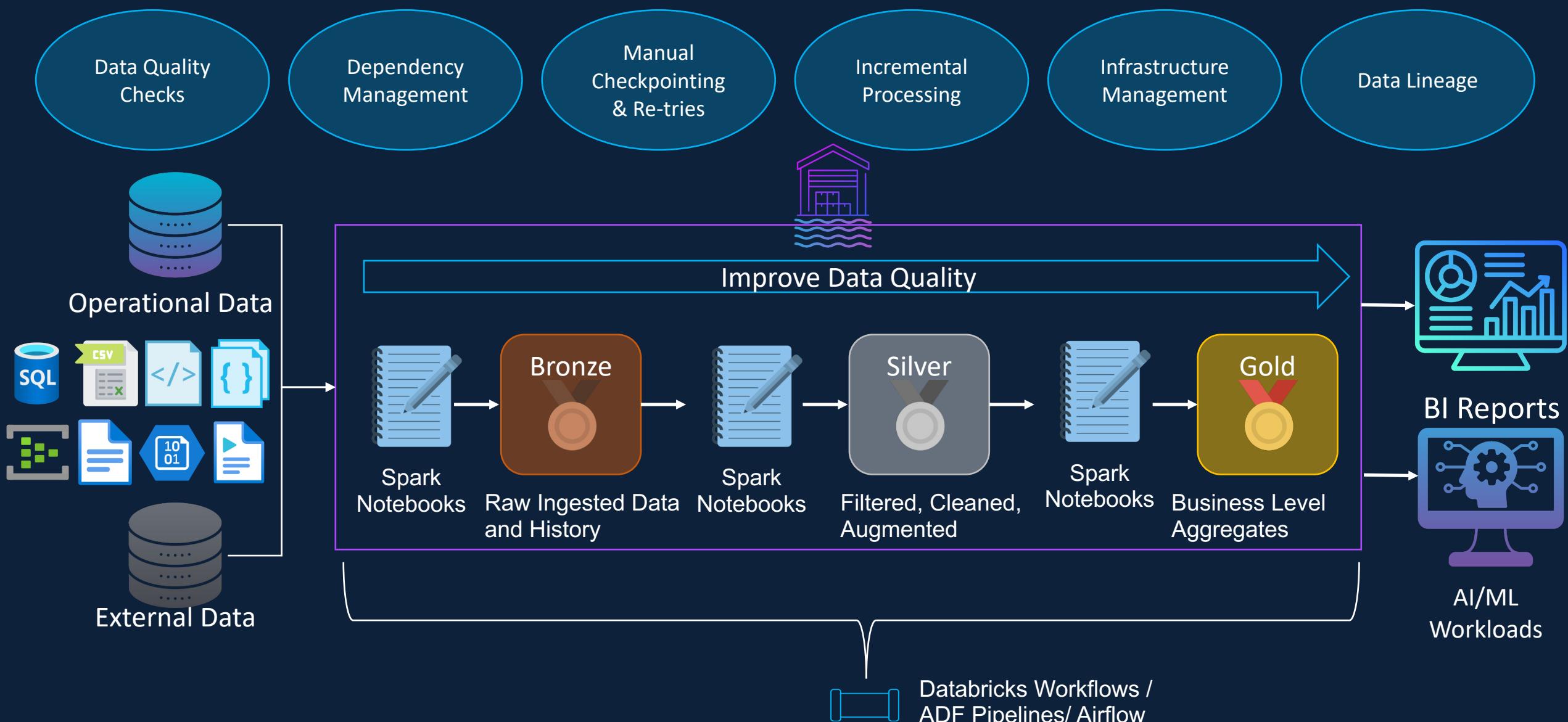
It handles both **streaming and batch workloads** with minimal manual intervention.

- *Databricks*

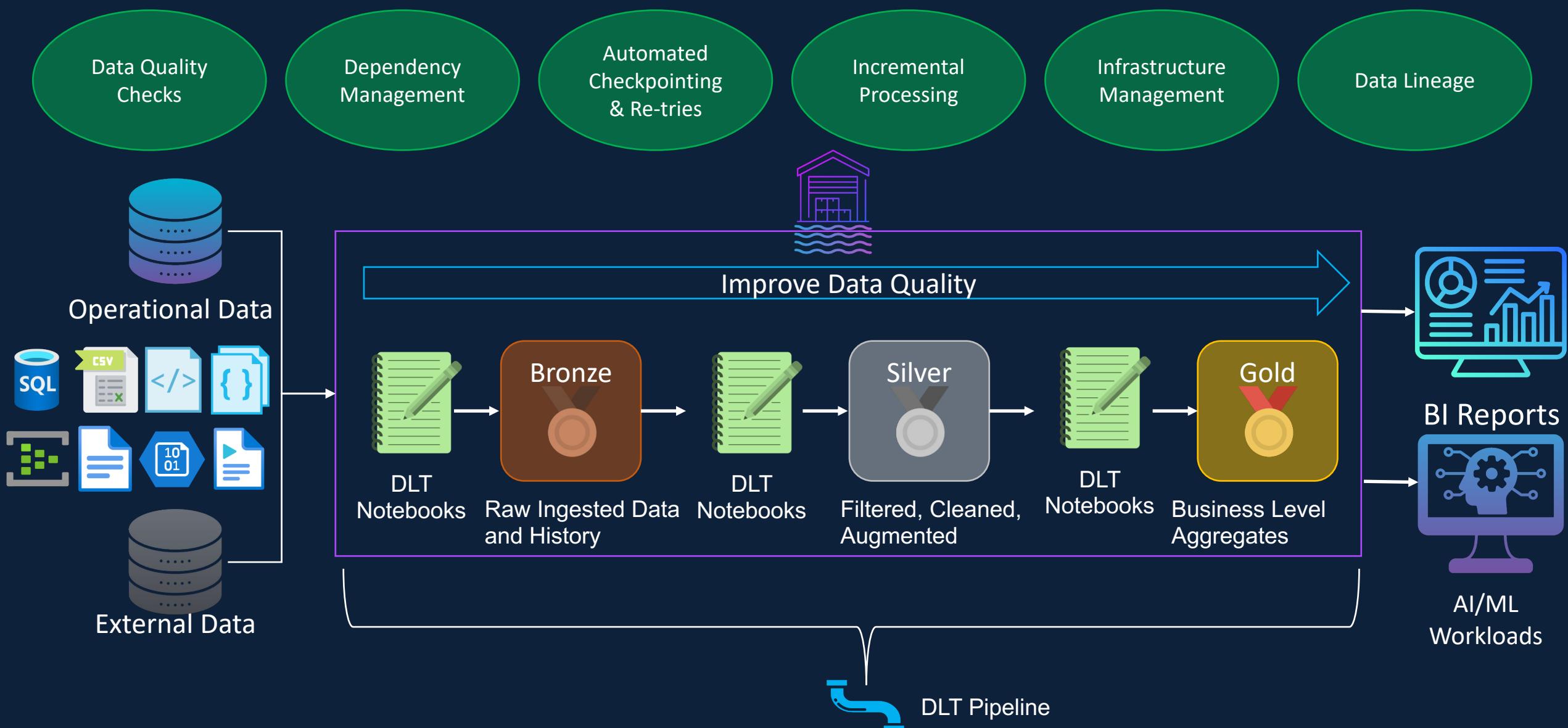
Delta Live Tables (DLT) - Overview



Delta Live Tables (DLT) - Overview



Delta Live Tables (DLT) - Overview



DLT Pipelines / Databricks Jobs

DLT Pipelines

Performs task orchestration with a pipeline

For managing a data flow

Databricks Jobs

For managing a workflow

Scheduling Tasks (including DLT)

Trigger Tasks based on Events (e.g. File Arrival)

Orchestrate ETL/ BI/ ML Tasks

Delta Live Tables (DLT) Architecture

Delta Live Tables (DLT) Architecture

Delta Live Tables (DLT)

User Interface

DLT Notebooks

SQL

Python



Data Transformations

Data Quality Expectations

DLT Pipelines



Hive Metastore

Unity Catalog

Delta Live Tables (DLT) Architecture

Delta Live Tables (DLT)

User Interface

DLT Notebooks

SQL

Python



Data Transformations

Data Quality Expectations

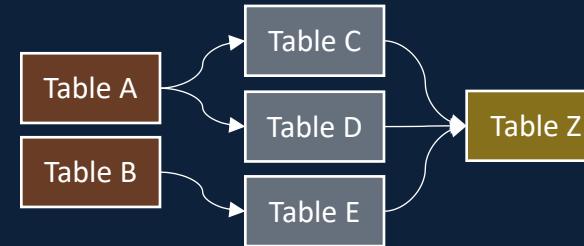
DLT Pipelines



Hive Metastore

Unity Catalog

DAG



Infrastructure



Metadata

Pipeline Status/ Health

Data Quality Checks

Data Lineage

Data Quality Checks

Dependency Management

Automated Checkpointing & Re-tries

Incremental Processing

Infrastructure Management

Data Lineage

Delta Live Tables - Programming

Delta Live Tables (DLT)

User Interface

DLT Notebooks

SQL

Python



Data Transformations

Data Quality Expectations

DLT Pipelines



Hive Metastore

Unity Catalog

Delta Live Tables - Programming

Delta Live Tables is a declarative ETL framework for building reliable, maintainable, and testable data processing pipelines.

Delta Live Tables - Programming

Imperative

How to do a task?

User provides step-by-step instructions

```
users_above_18 = []
for user in users:
    if user.age > 18:
        users_above_18.append(user)
```

Declarative

What should be done?

System decides how to do it

```
SELECT * FROM users WHERE age > 18;
```

Delta Live Tables - Programming

Declarative

What should be done?

System decides how to do it

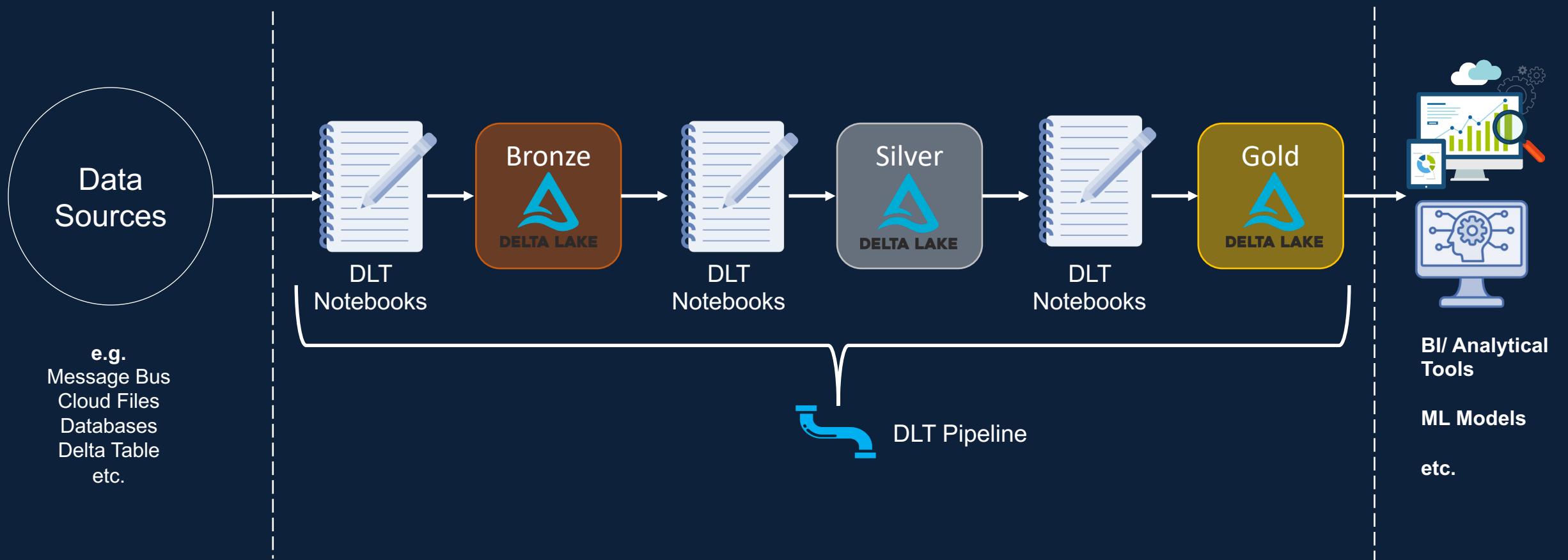
```
SELECT * FROM users WHERE age > 18;
```

No need to handle checkpoints, fault tolerance, dependencies etc

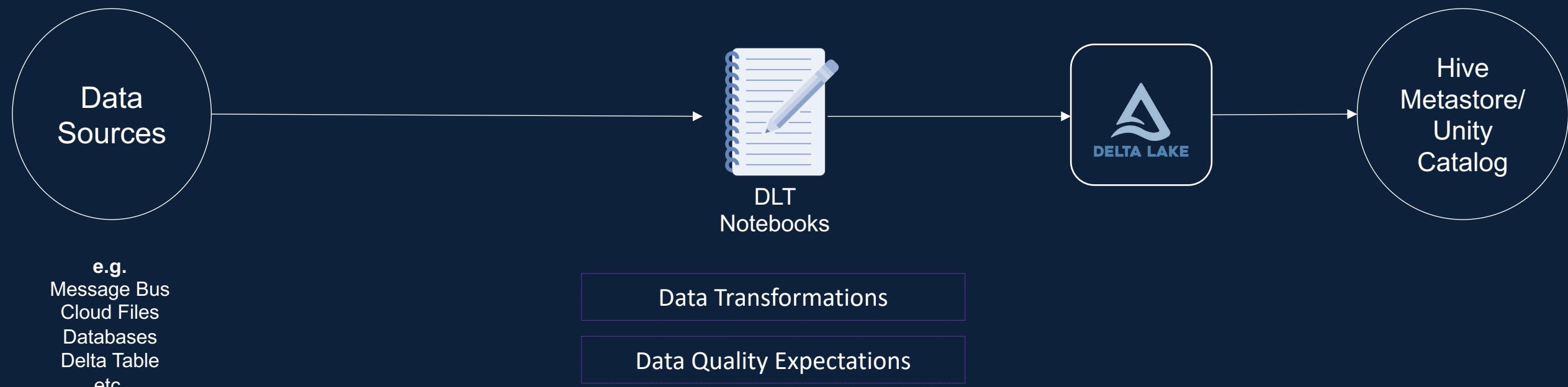
No need to write complex logic such as CDC, Schema Evolution etc.

Better Optimization

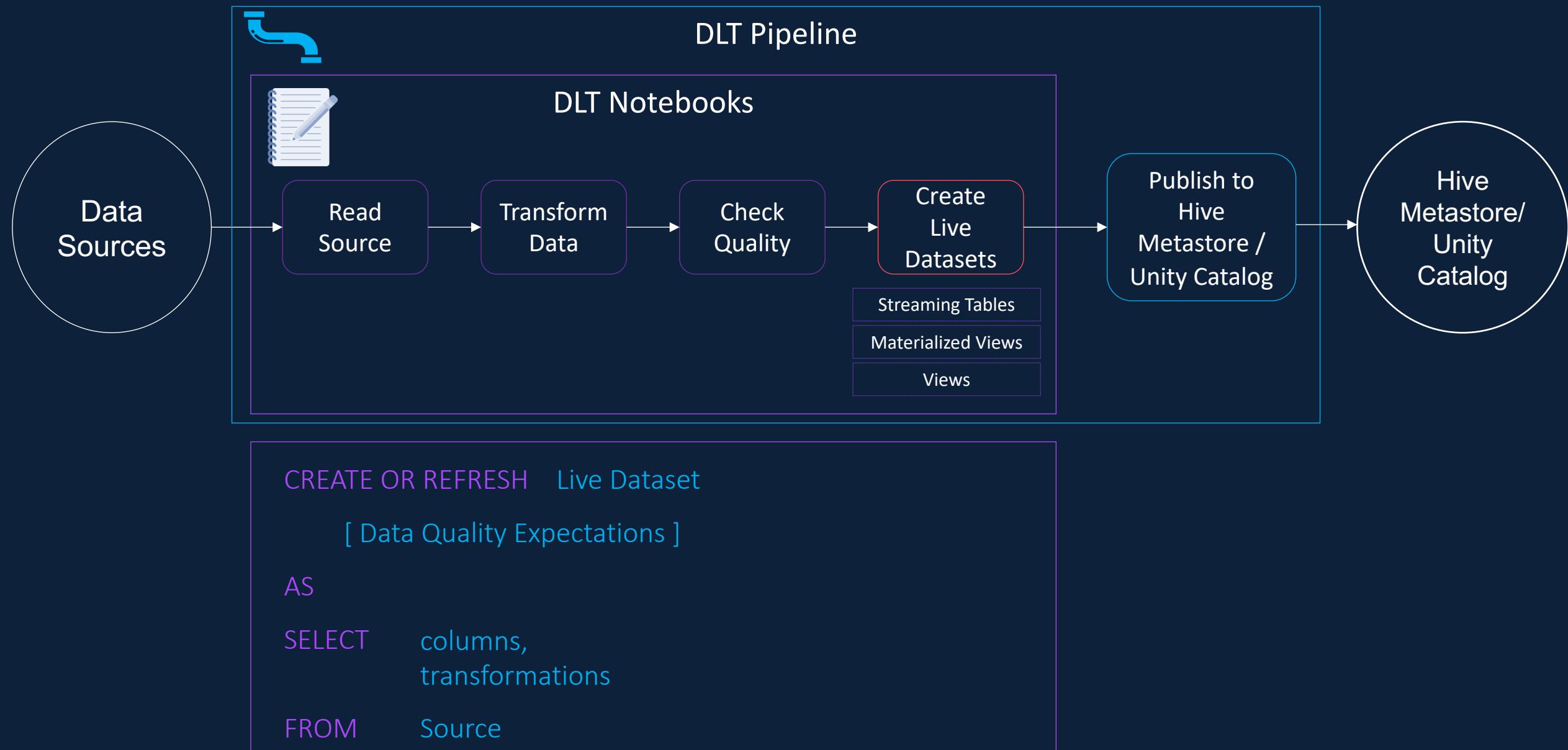
Delta Live Tables - Programming



Delta Live Tables - Programming



Delta Live Tables - Programming



Delta Live Tables - Programming



Streaming Tables

Delta Table to which streams write data

Reads from Eventhub, Kafka, Cloud Files, Delta Tables etc.

Offers exactly once guarantees

Incremental Data Ingestion Workloads

Low latency transformations

Massive amount of data

DML operations allowed



Materialized Views

Delta Table created from the result of a query

Full Refresh Data Ingestion Workloads

Build aggregate tables for reporting

Improve latency of BI reports

Data Transformation Workloads

No DML operations allowed - Change the query instead.



Views

No physical storage of the data

Scope is limited to the pipeline

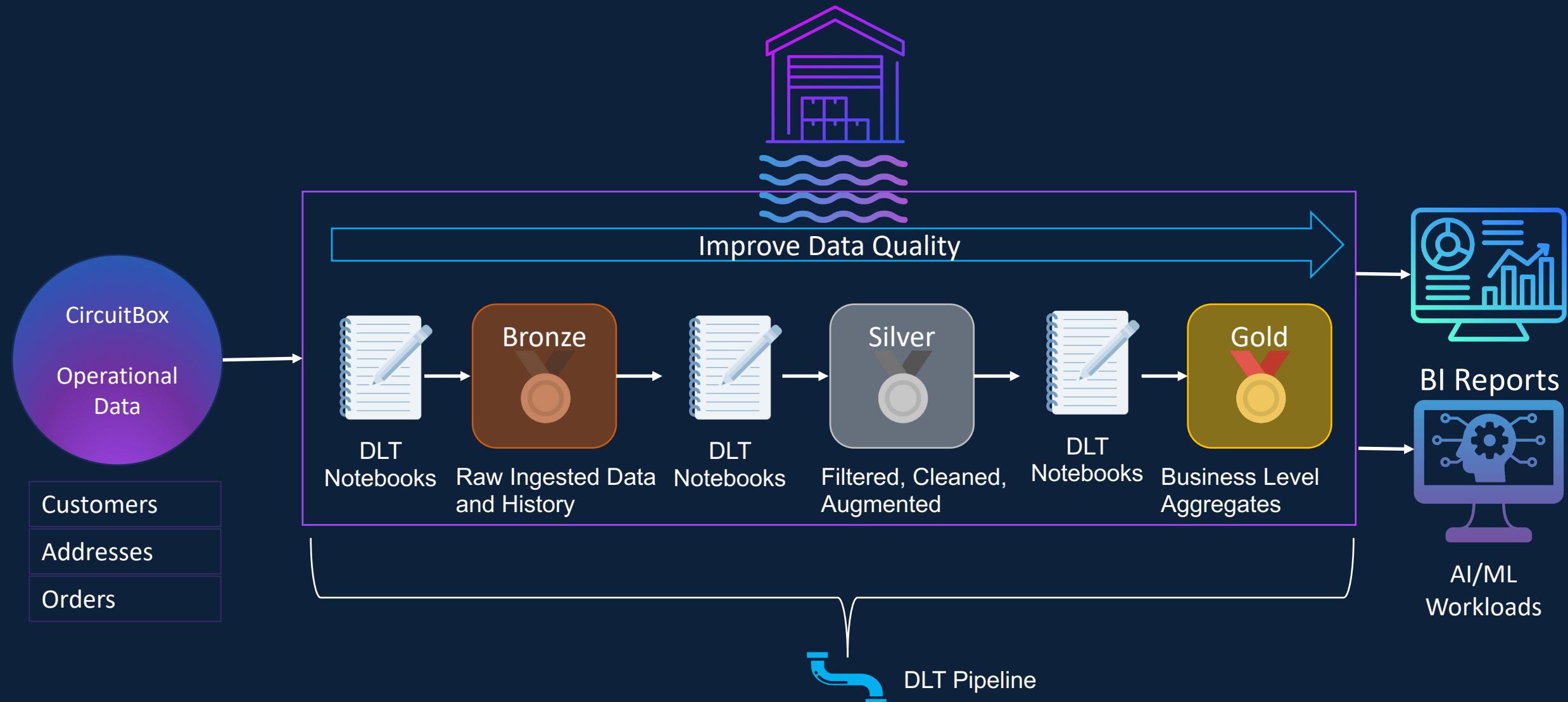
Cannot be published to Hive Metastore/Unity Catalog

Store intermediate results to reduce complexity

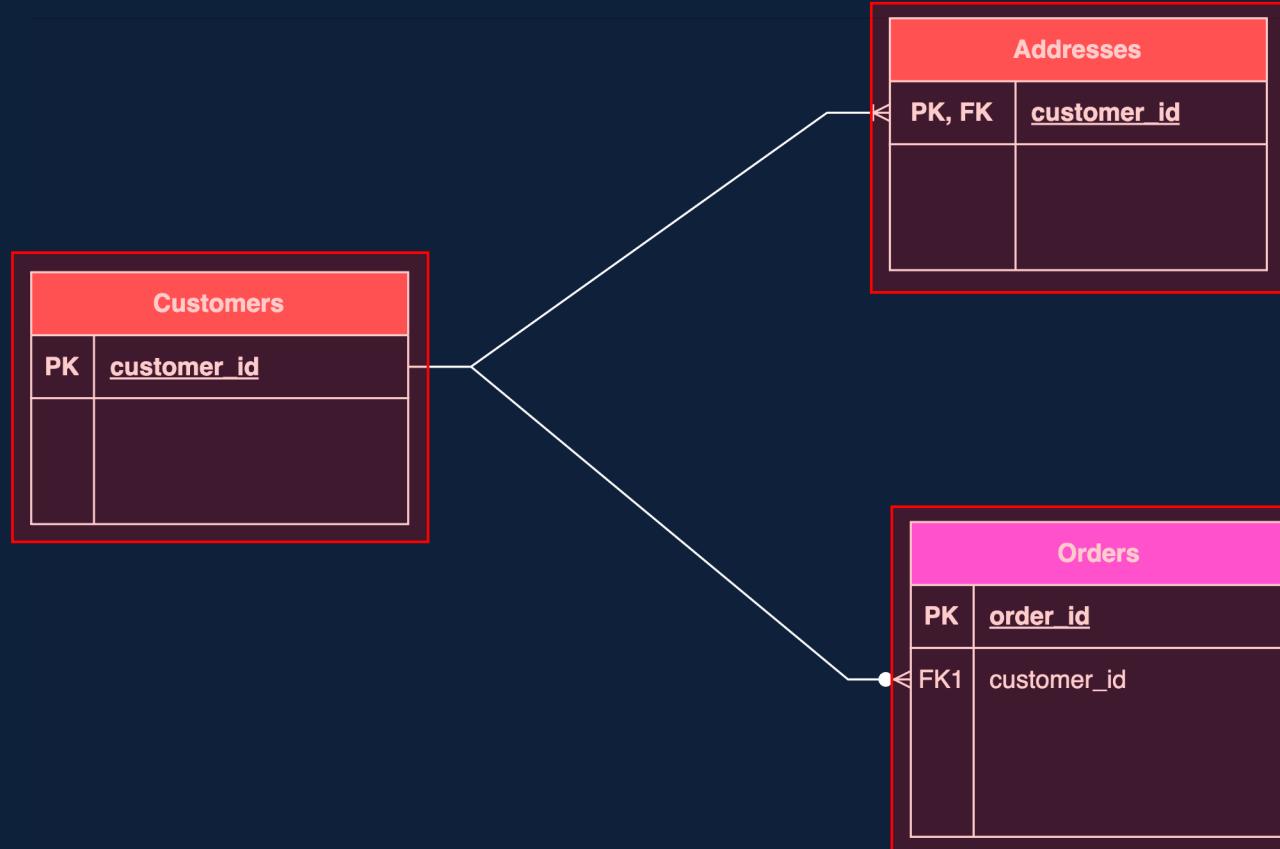
To enforce data quality constraints

Delta Live Tables (DLT) Project - CircuitBox Introduction

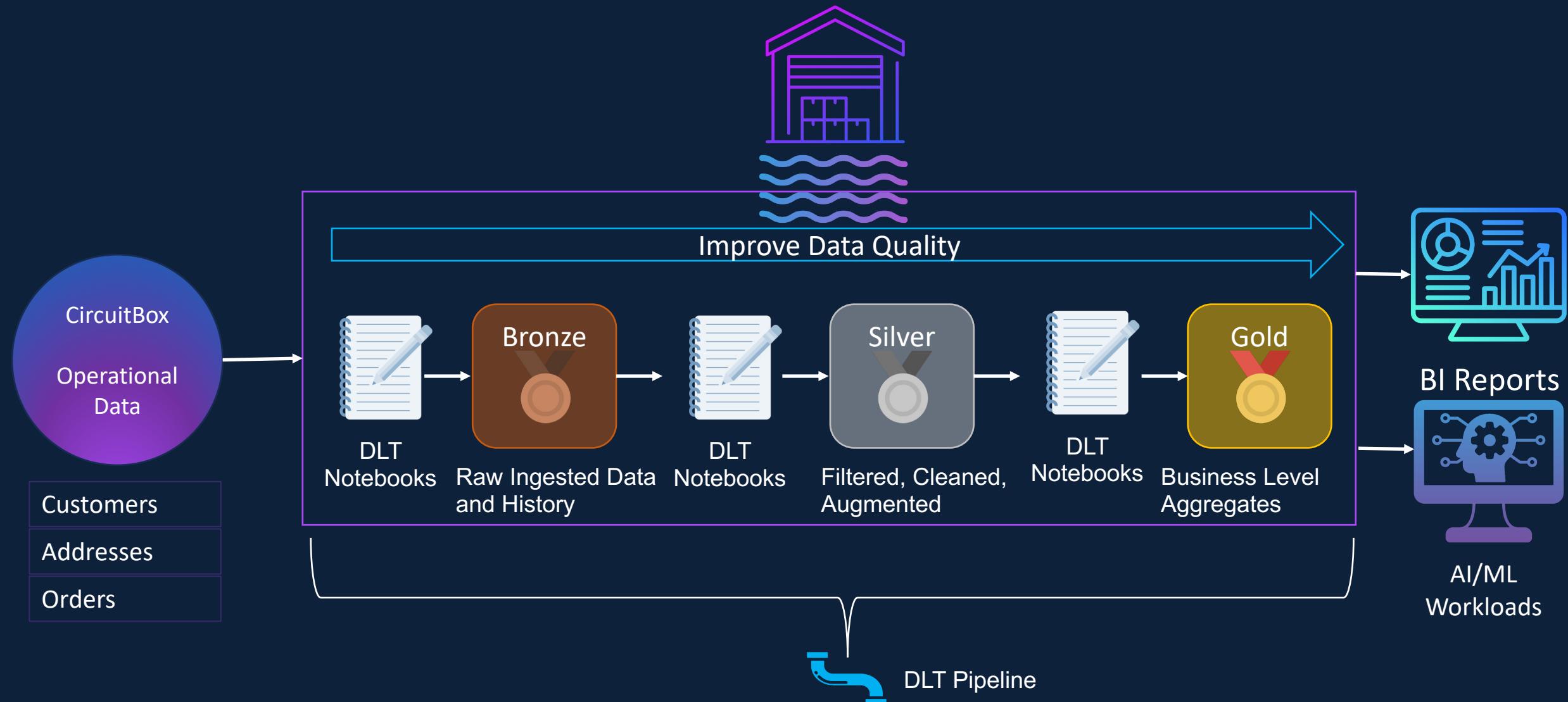
Delta Live Tables (DLT) Project - CircuitBox



Delta Live Tables (DLT) Project - CircuitBox



Delta Live Tables (DLT) Project - CircuitBox



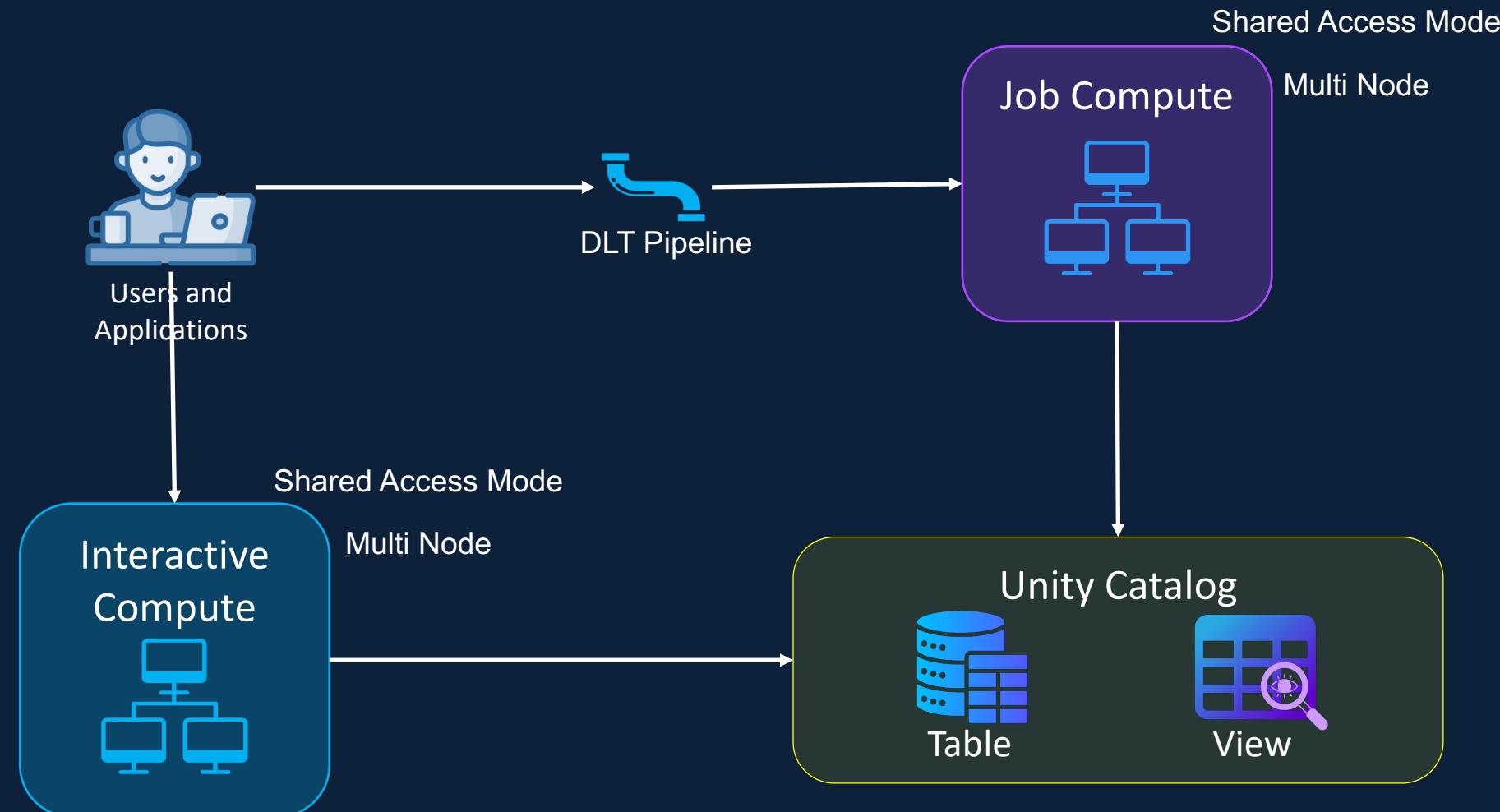
Delta Live Tables (DLT) Project - CircuitBox



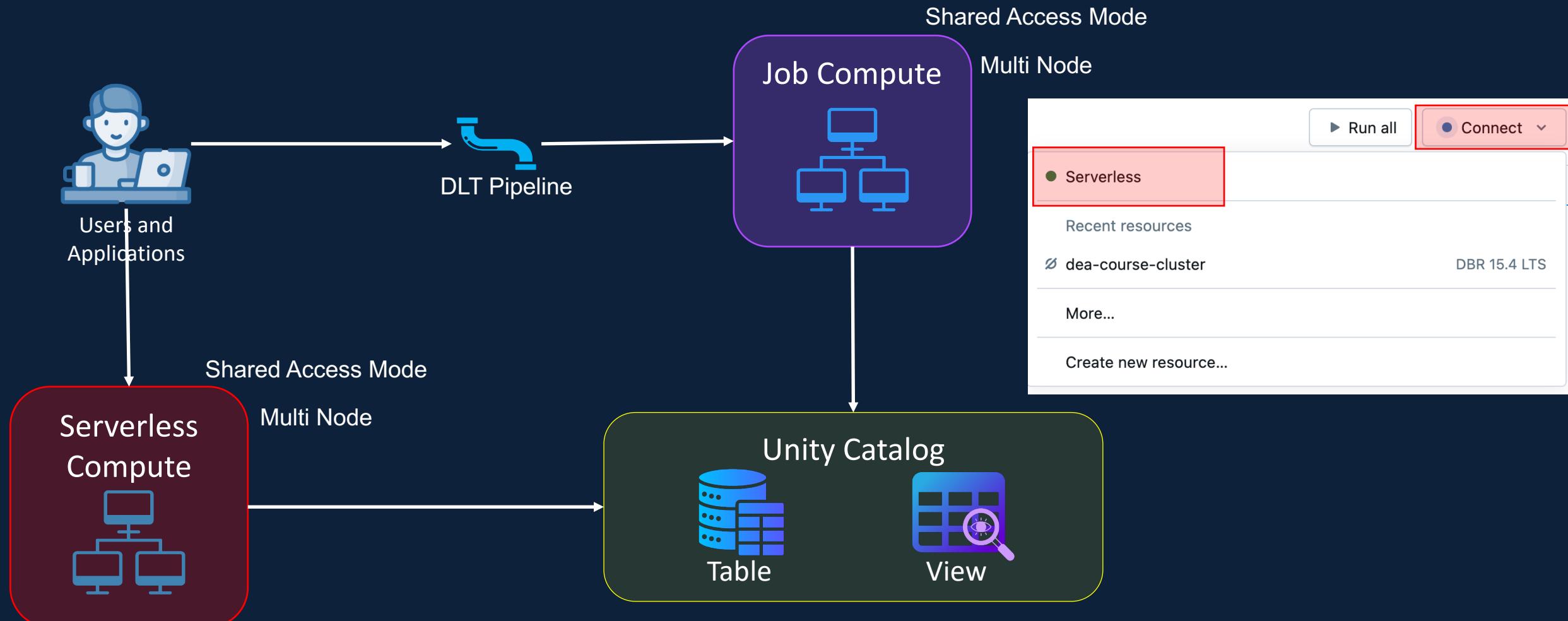
Compute/ Cluster Configuration

DLT Project

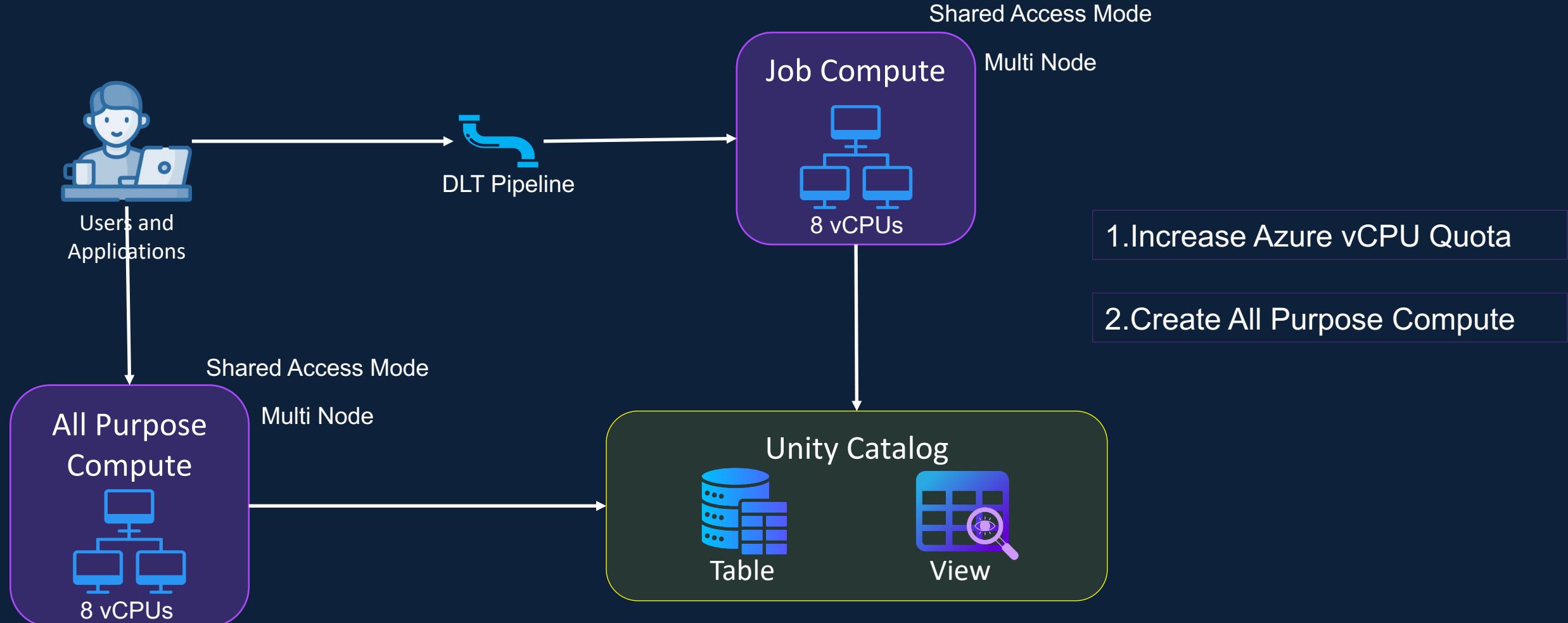
Compute/ Cluster Configuration



Compute/ Cluster Configuration

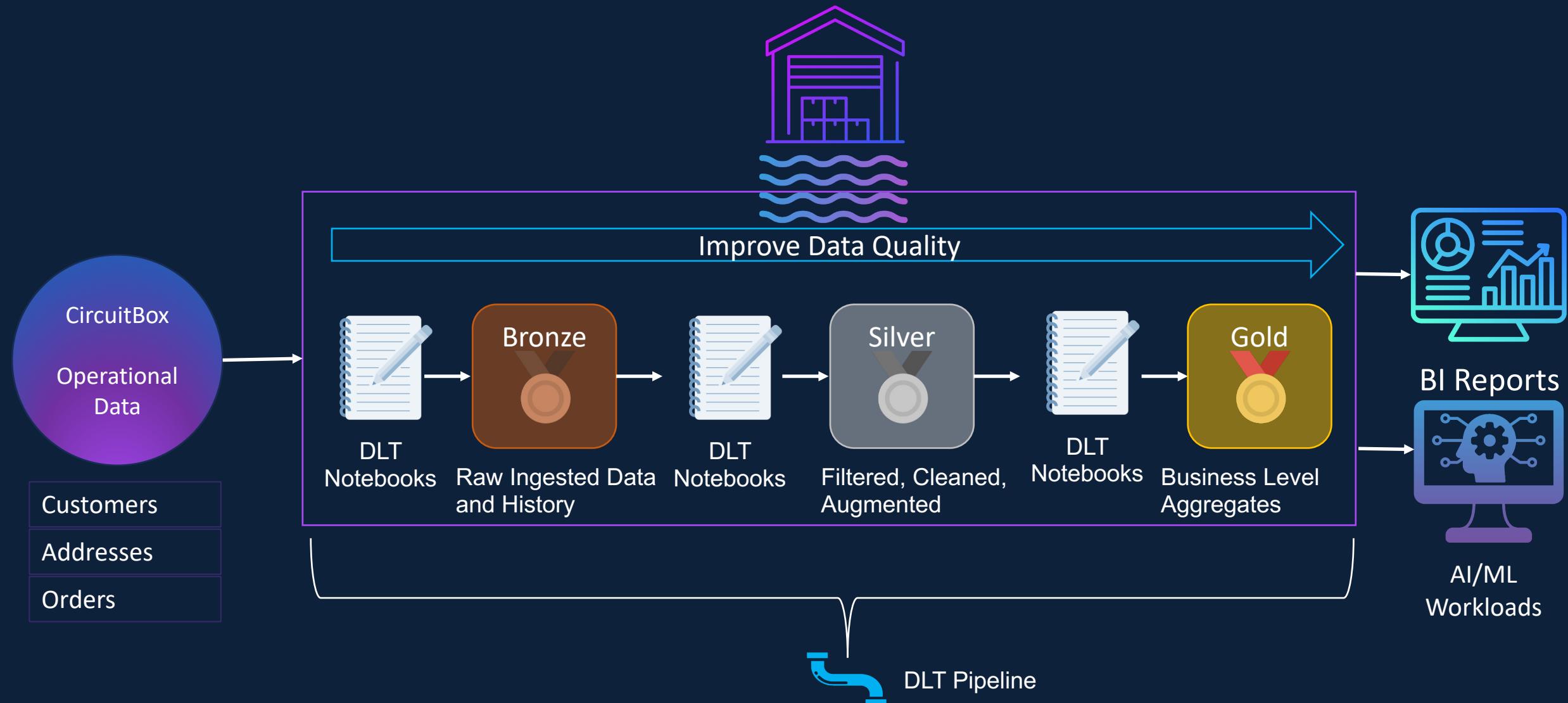


Compute/ Cluster Configuration

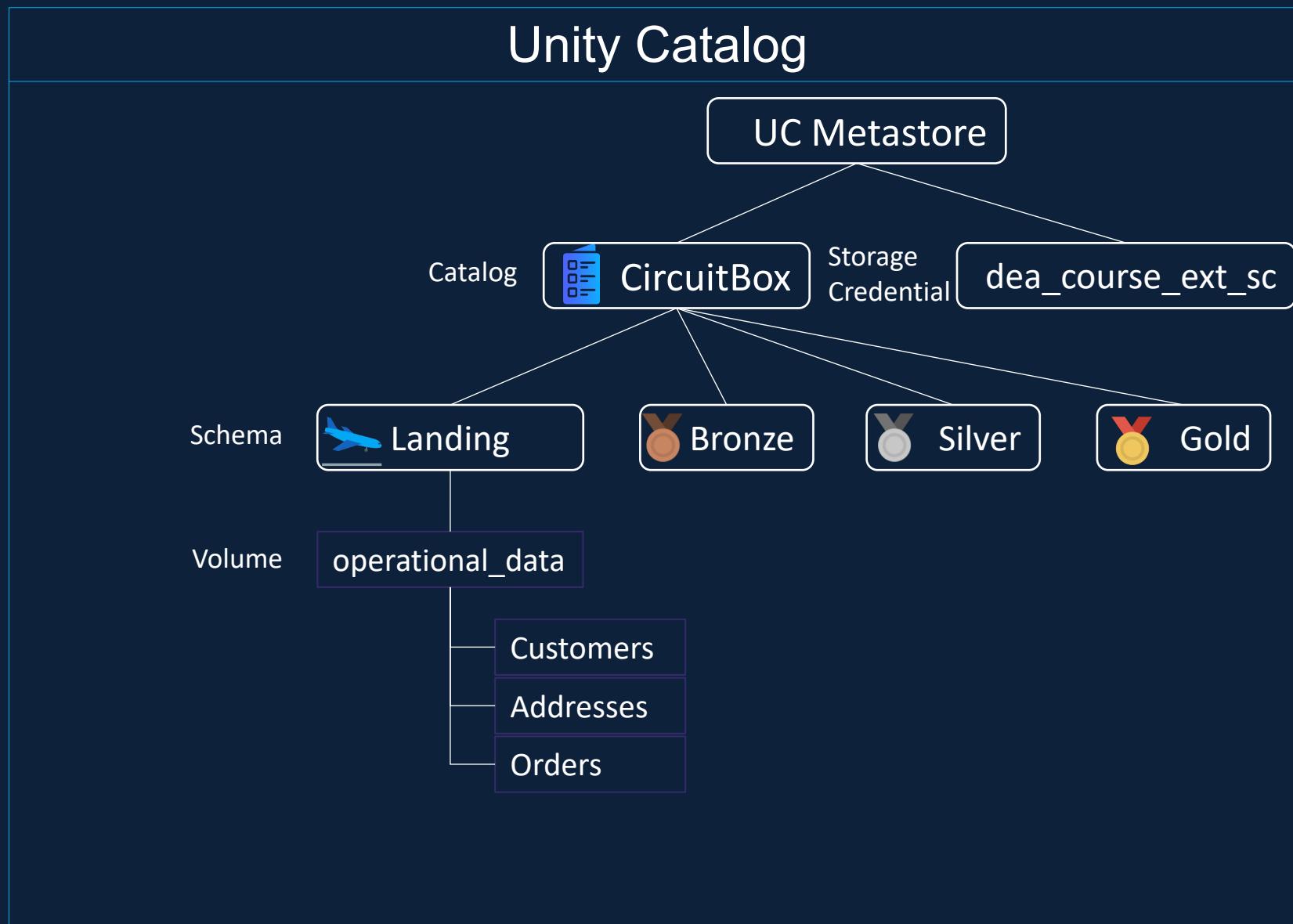


Set-up Project Environment

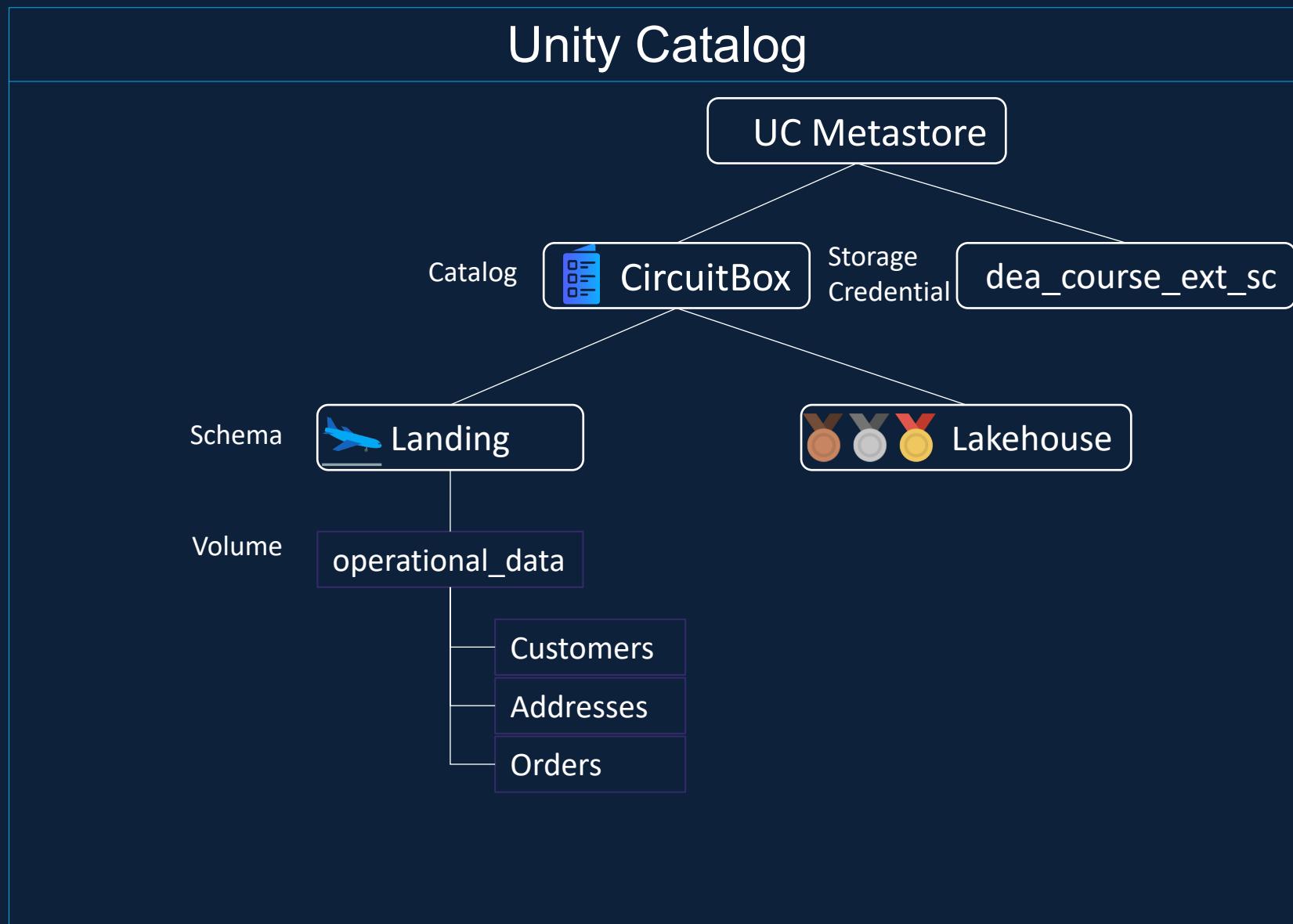
Delta Live Tables (DLT) Project - CircuitBox



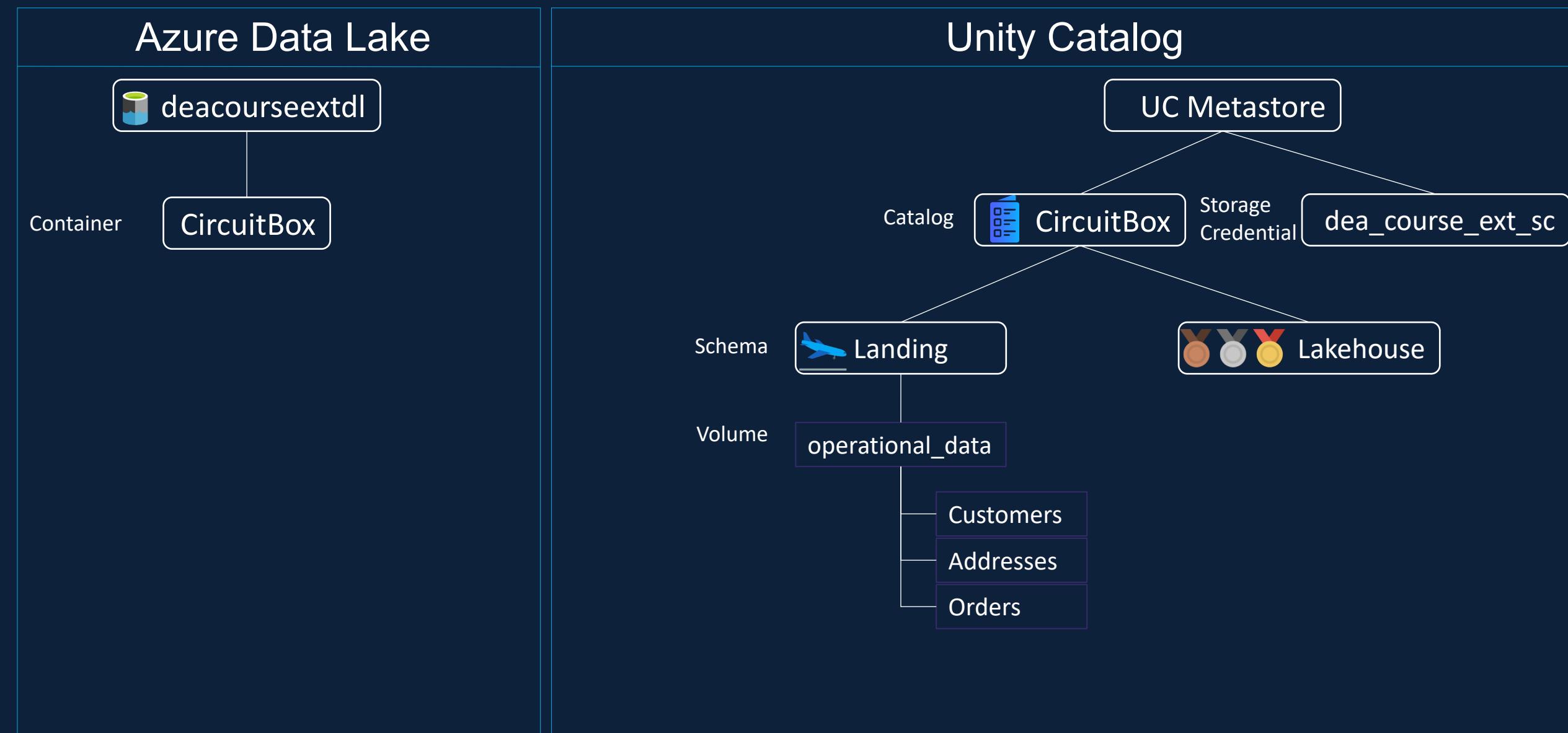
Data Architecture - CircuitBox



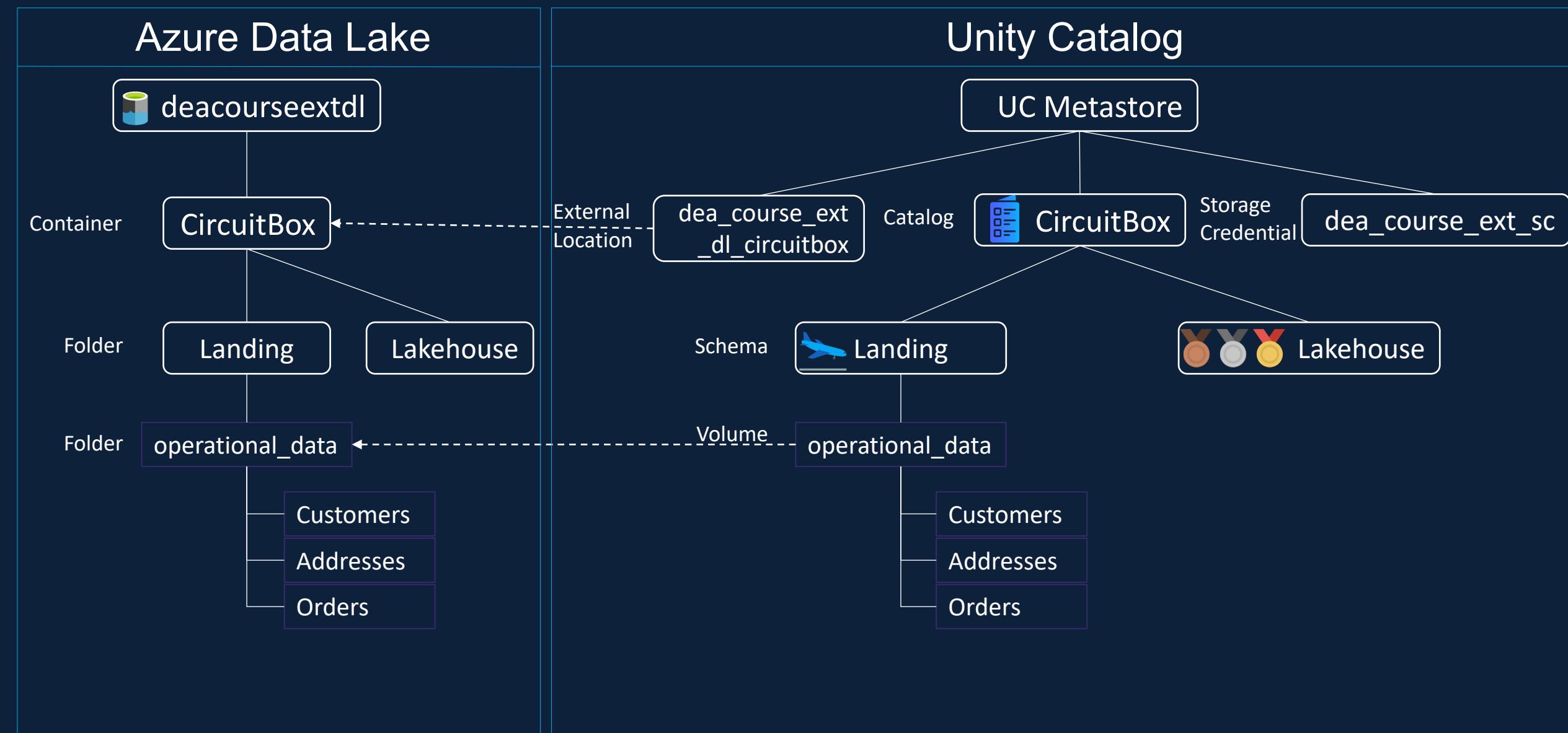
Data Architecture - CircuitBox



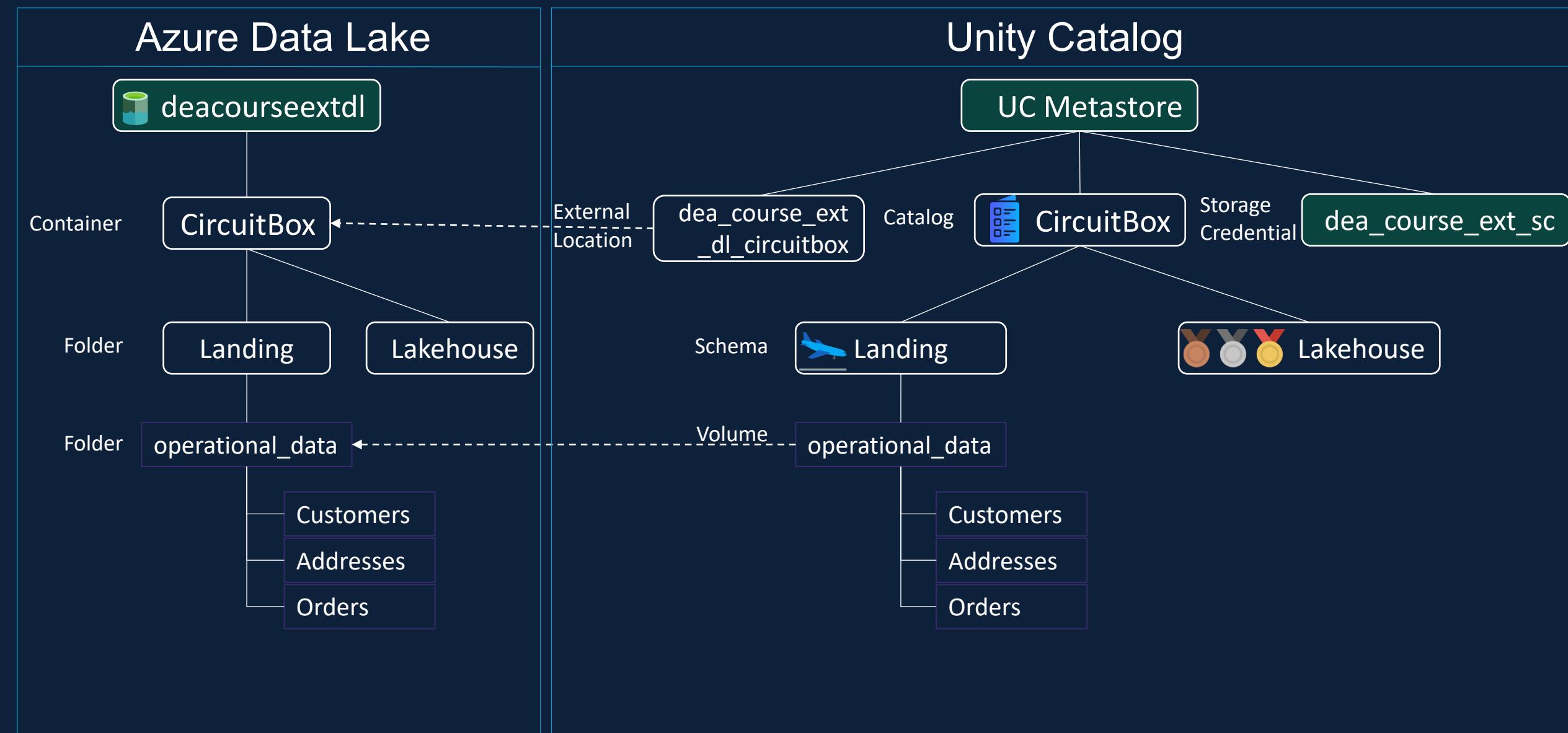
Data Architecture - CircuitBox



Data Architecture - CircuitBox



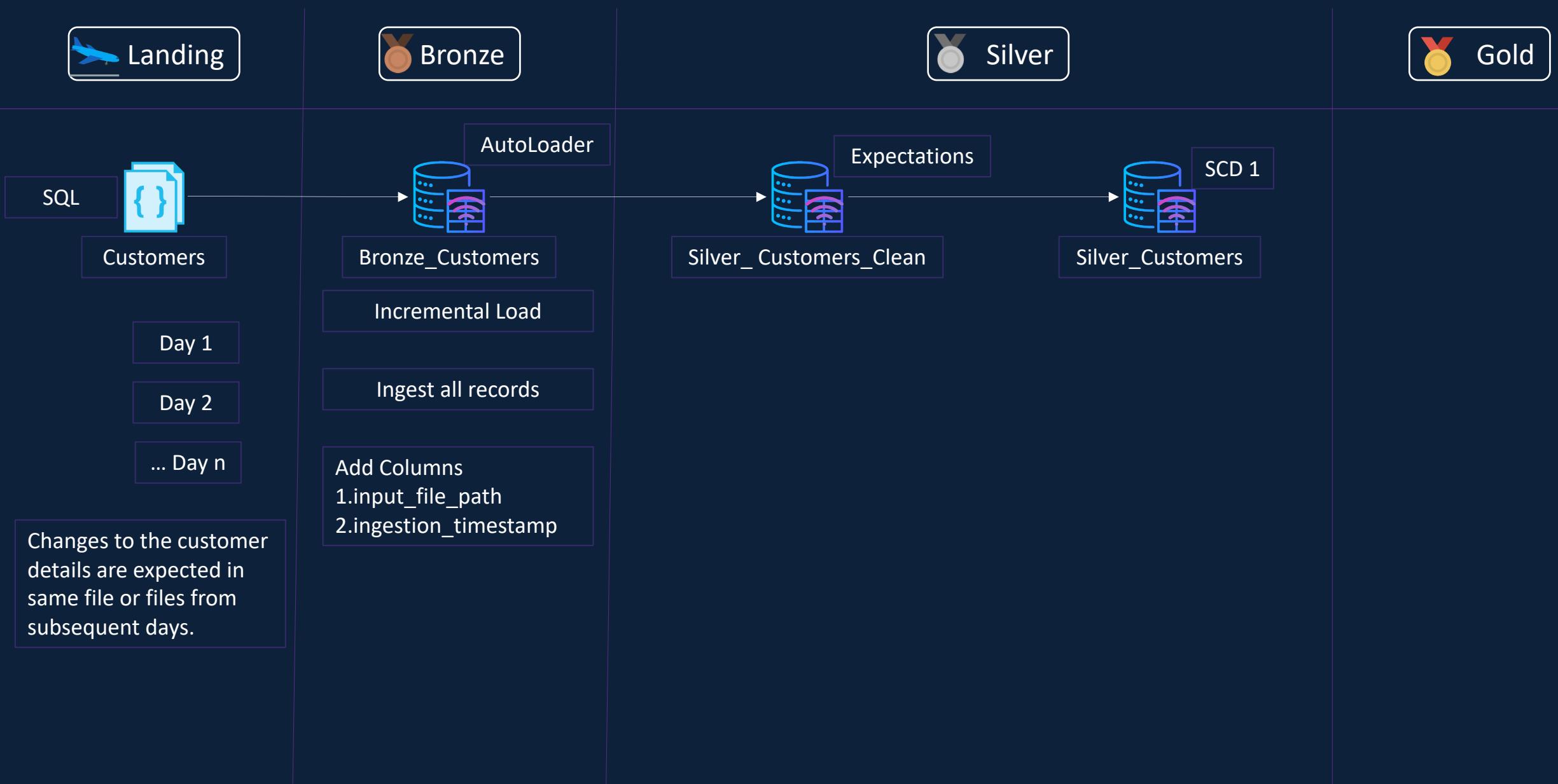
Data Architecture - CircuitBox



Delta Live Tables (DLT) Project - CircuitBox



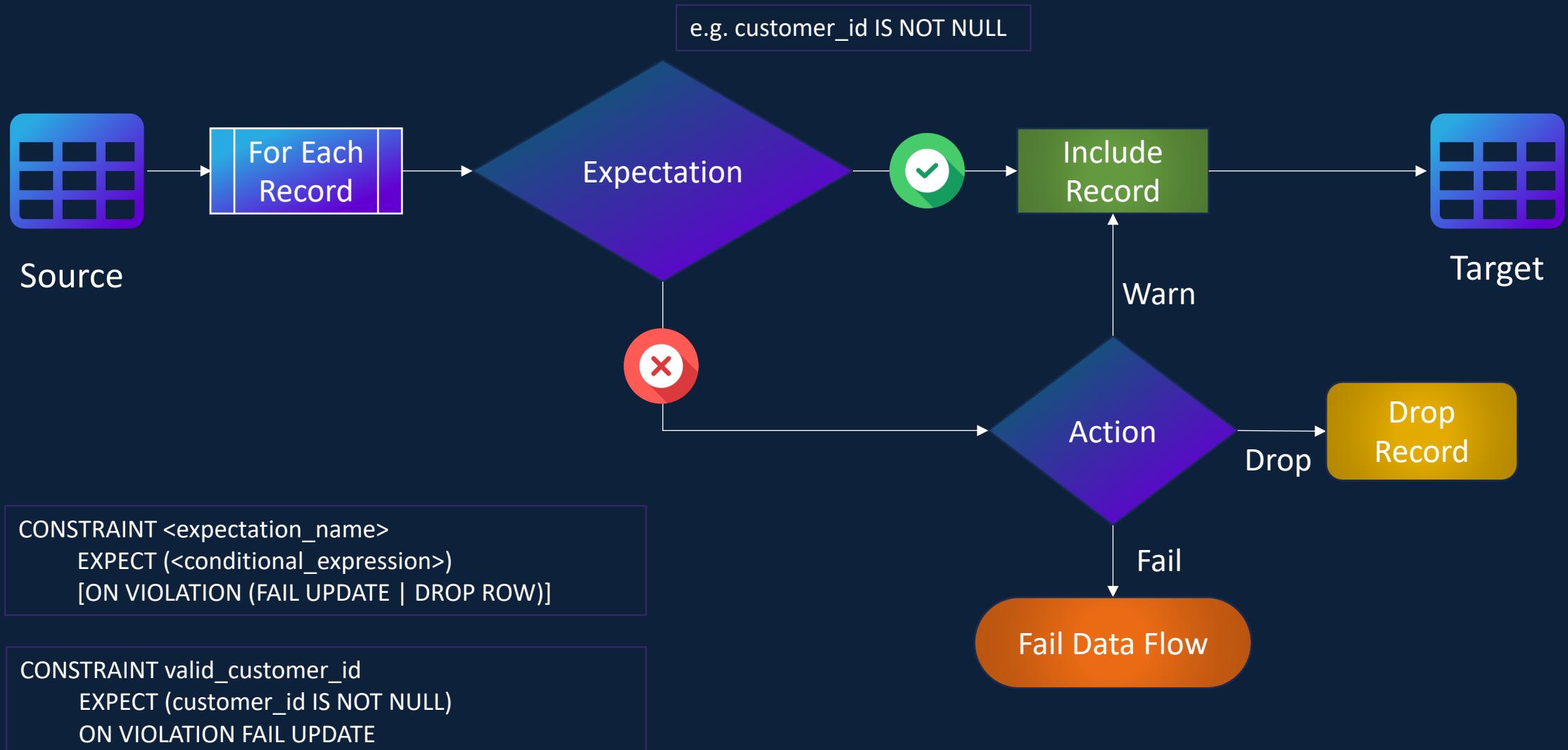
CircuitBox – Process Customers Data



CircuitBox – Process Customers Data



Delta Live Tables (DLT) - Expectations

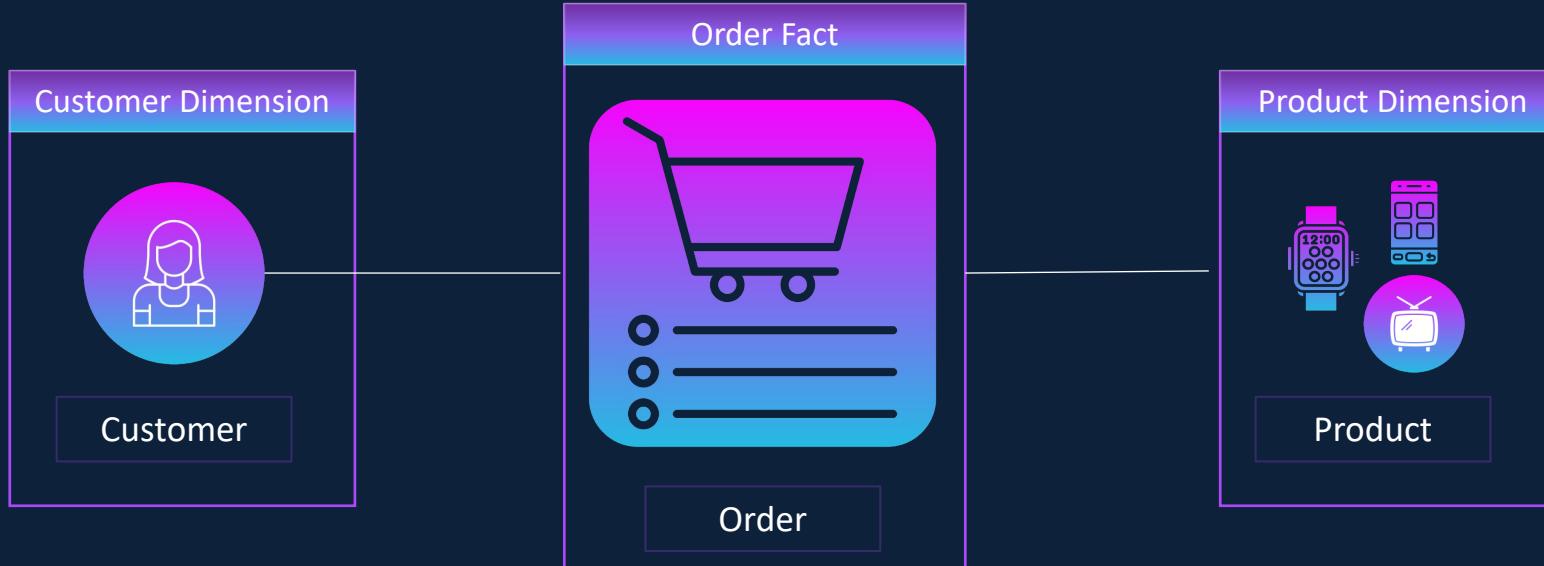


CircuitBox – Process Customers Data



Slowly Changing Dimensions

Slowly Changing Dimensions

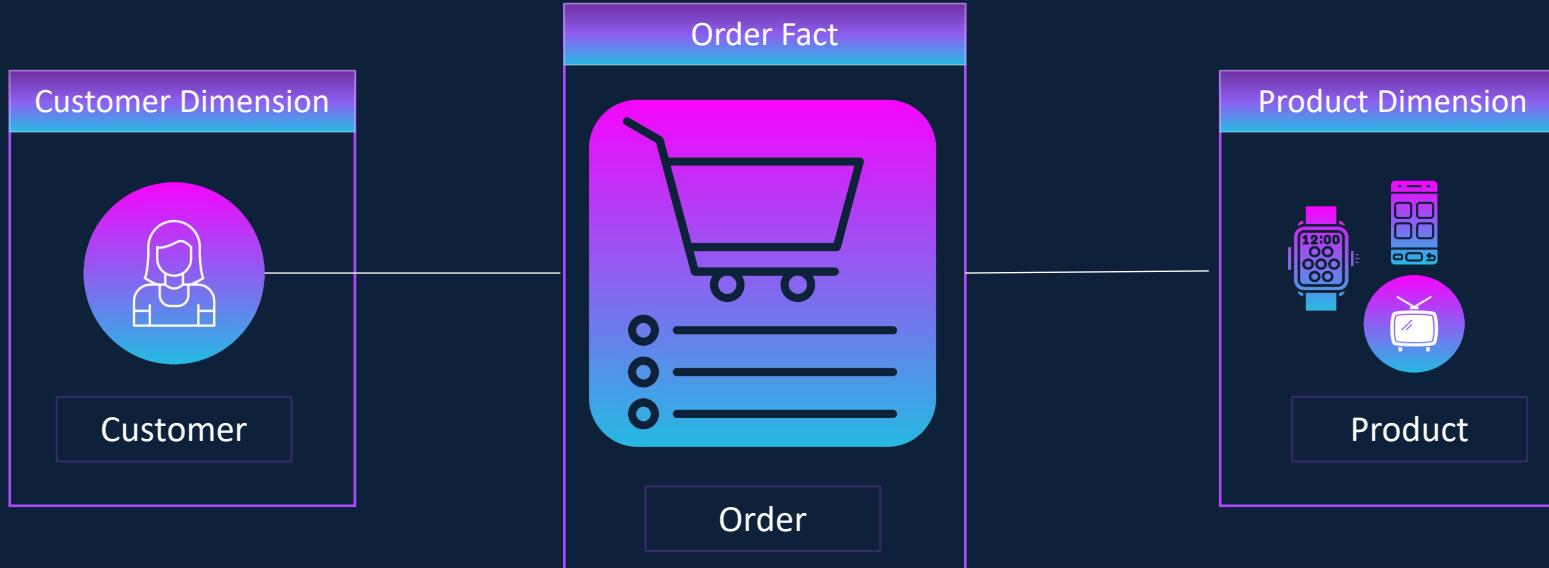


SCD Type 1 – Overwrite the data (No History Maintained)

Before	customer_id	Customer_name	telephone
	1001	Diana Miller	+1 212 555 1234

After	customer_id	Customer_name	telephone
	1001	Diana Miller	+65 6123 4567

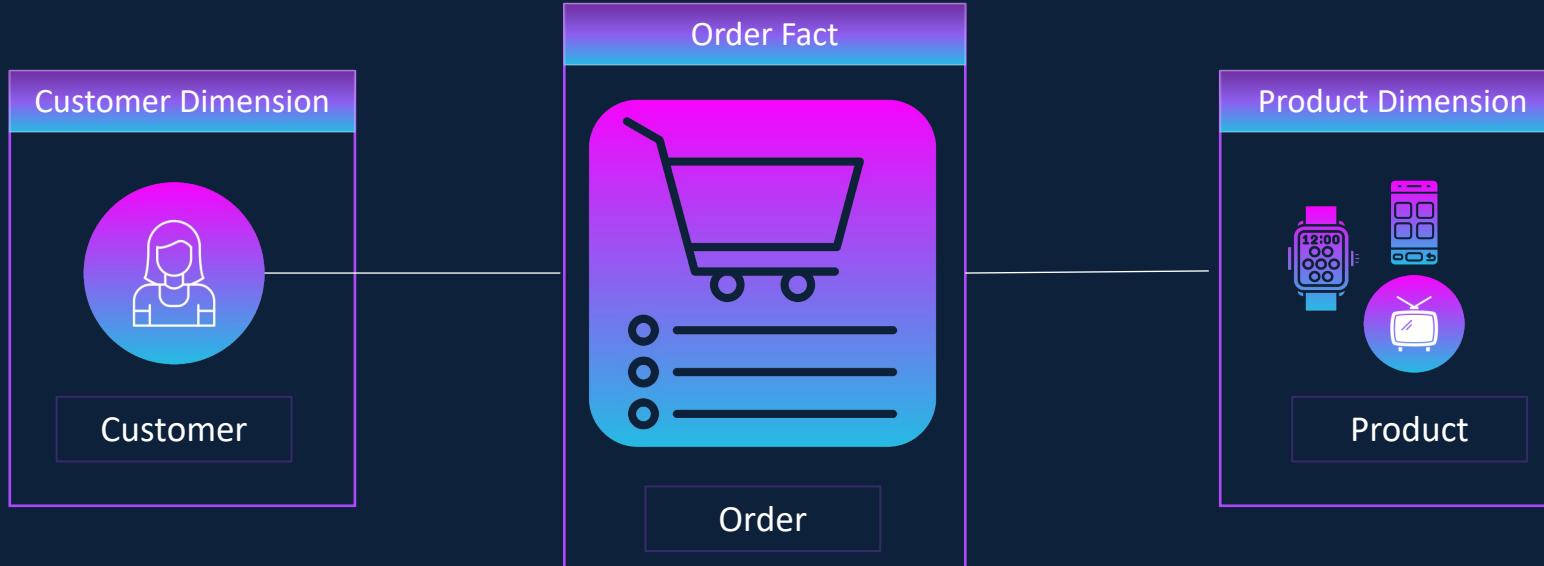
Slowly Changing Dimensions



SCD Type 2 – Full History Maintained

Before	customer_id	Customer_name	telephone	start_date	end_date
	1001	Diana Miller	+1 212 555 1234	2021-01-01	
After					
After	customer_id	Customer_name	telephone	start_date	end_date
	1001	Diana Miller	+1 212 555 1234	2021-01-01	2025-01-26
	1001	Diana Miller	+65 6123 4567	2025-01-26	

Slowly Changing Dimensions



SCD Type 3 – Only the most recent change is tracked

Before	customer_id	Customer_name	telephone	prev_telephone
	1001	Diana Miller	+1 212 555 1234	

After	customer_id	Customer_name	telephone	prev_telephone
	1001	Diana Miller	+65 6123 4567	+1 212 555 1234

CircuitBox – Process Customers Data



APPLY CHANGES API

```
CREATE OR REFRESH STREAMING TABLE <target_table>;
```

```
APPLY CHANGES INTO <target_table>
  FROM <source_table>
  KEYS <columns>
  SEQUENCE BY <columns>
  STORED AS <SCD TYPE 1 | SCD TYPE 2>
```



```
CREATE OR REFRESH STREAMING TABLE silver_customers;
```

```
APPLY CHANGES INTO LIVE.silver_customers
  FROM STREAM(LIVE.silver_customers_clean)
  KEYS (customer_id)
  SEQUENCE BY created_date
  STORED AS SCD TYPE 1;
```

APPLY CHANGES API

```
APPLY CHANGES INTO LIVE.table_name
```

```
    FROM source
```

```
    KEYS (keys)
```

```
    [IGNORE NULL UPDATES]
```

```
    [APPLY AS DELETE WHEN condition]
```

```
    [APPLY AS TRUNCATE WHEN condition]
```

```
    SEQUENCE BY orderByColumn
```

```
    [COLUMNS {columnList | * EXCEPT (exceptColumnList)}]
```

```
    [STORED AS {SCD TYPE 1 | SCD TYPE 2}]
```

```
    [TRACK HISTORY ON {columnList | * EXCEPT (exceptColumnList)}]
```

Delta Live Tables (DLT) Project - CircuitBox



CircuitBox – Process Addresses Data



DLT Programming using Python

PySpark

Read Source Data

```
input_df = spark.read....
```

```
input_df = spark.readStream...
```

Apply Transformations

```
transformed_df = input_df.transform(...)
```

Write Transformed Data

```
transformed_df.write...
```

Delta Live Tables (DLT)

```
Import dlt
```

```
@dlt.table ()
```

```
def <function_name>():
```

Read Source Data

Apply Transformations

```
return <dataframe>
```

DLT Programming using Python

Delta Live Tables (DLT)

Import dlt

```
@dlt.table ()  
def <function_name>():  
    Read Source Data  
    Apply Transformations  
    return <dataframe>
```

Example

Import dlt

```
@dlt.table ()  
def bronze_addresses():  
    return(  
        spark.readStream  
            .format("cloudFiles")  
            .option("cloudFiles.format", "csv")  
            .option("cloudFiles.inferColumnTypes", "true")  
            .load("/<volume_path>/addresses/")  
    )
```

DLT Programming using Python

Delta Live Tables (DLT)

Import dlt

```
@dlt.table ()  
def <function_name>():  
    Read Source Data  
    Apply Transformations  
    return <dataframe>
```

Example

Import dlt

```
@dlt.table (  
    name = 'bronze_addresses',  
    comment = 'raw addresses data'),  
    table_properties = {'quality' : 'bronze'}  
)  
  
def bronze_addresses():  
    return(  
        spark.readStream  
            .format("cloudFiles")  
            .option("cloudFiles.format", "csv")  
            .option("cloudFiles.inferColumnTypes", "true")  
            .load("/<volume_path>/addresses/")  
)
```

CircuitBox – Process Addresses Data



DLT Programming using Python



Stream Read

dlt.read_stream()

spark.readStream.table()

Static Read

dlt.read ()

spark.read.table()

DLT Programming using Python

Stream Read



```
spark.readStream.table()
```

Static Read

```
spark.read.table()
```

Example

Import dlt

```
@dlt.table ()
```

```
def silver_addresses_clean():
    return(
        spark.readStream.table("LIVE.bronze_addresses")
    )
```

DLT Programming using Python

Warning

```
@dlt.expect
```

```
@dlt.expect("valid_postcode",  
            "length(postcode) = 5")
```

Drop

```
@dlt.expect_or_drop
```

```
@dlt.expect_or_drop("valid_address_line_1",  
                     "address_line_1 IS NOT NULL")
```

Fail

```
@dlt.expect_or_fail
```

```
@dlt.expect_or_fail("valid_customer_id",  
                     "customer_id IS NOT NULL")
```

DLT Programming using Python

Warning

```
@dlt.expect_all
```

Drop

```
@dlt.expect_all_or_drop
```

Fail

```
@dlt.expect_all_or_fail
```

```
Valid_records = {"valid_customer_id" : "customer_id IS NOT NULL",
                 "valid_name" : " name IS NOT NULL",
                 "valid_postcode" : "postcode IS NOT NULL"}
```

```
@dlt.expect_all_or_drop(valid_records)
```

DLT Programming using Python

Warning

```
@dlt.expect  
@dlt.expect_all
```

Drop

```
@dlt.expect_or_drop  
@dlt.expect_all_or_drop
```

Fail

```
@dlt.expect_or_fail  
@dlt.expect_all_or_fail
```

Example

Import dlt

```
@dlt.table()
```

```
def silver_addresses_clean():  
    return(  
        spark.readStream.table("LIVE.bronze_addresses")  
    )
```

DLT Programming using Python

Warning

@dlt.expect

@dlt.expect_all

Drop

@dlt.expect_or_drop

@dlt.expect_all_or_drop

Fail

@dlt.expect_or_fail

@dlt.expect_all_or_fail

Example

Import dlt

@dlt.table()

@dlt.expect_or_fail("customer_id", "customer_id IS NOT NULL")

@dlt.expect_or_drop("address_line_1", "address_line_1 IS NOT NULL")

@dlt.expect("postcode", "length(postcode) = 5")

```
def silver_addresses_clean():
    return(
        spark.readStream.table("LIVE.bronze_addresses")
    )
```

CircuitBox – Process Addresses Data



DLT Programming using Python

Create Streaming Table

```
create_streaming_table(  
    name = "<table-name>",  
    comment = "<comment>",  
    spark_conf={"<key>" : "<value>", "<key>" : "<value>"},  
    table_properties={"<key>" : "<value>", "<key>" : "<value>"},  
    partition_cols=["<partition-column>", "<partition-column>"],  
    cluster_by = ["<clustering-column>", "<clustering-column>"],  
    path="<storage-location-path>",  
    schema="schema-definition",  
    expect_all = {"<key>" : "<value>", "<key>" : "<value>"},  
    expect_all_or_drop = {"<key>" : "<value>", "<key>" : "<value>"},  
    expect_all_or_fail = {"<key>" : "<value>", "<key>" : "<value>"},  
    row_filter = "row-filter-clause"  
)
```

Apply Changes API

```
apply_changes(  
    target = "<target-table>",  
    source = "<data-source>",  
    keys = ["key1", "key2", "keyN"],  
    sequence_by = "<sequence-column>",  
    ignore_null_updates = False,  
    apply_as_deletes = None,  
    apply_as_truncates = None,  
    column_list = None,  
    except_column_list = None,  
    stored_as_scd_type = <type>,  
    track_history_column_list = None,  
    track_history_except_column_list = None  
)
```

Delta Live Tables (DLT) Project - CircuitBox



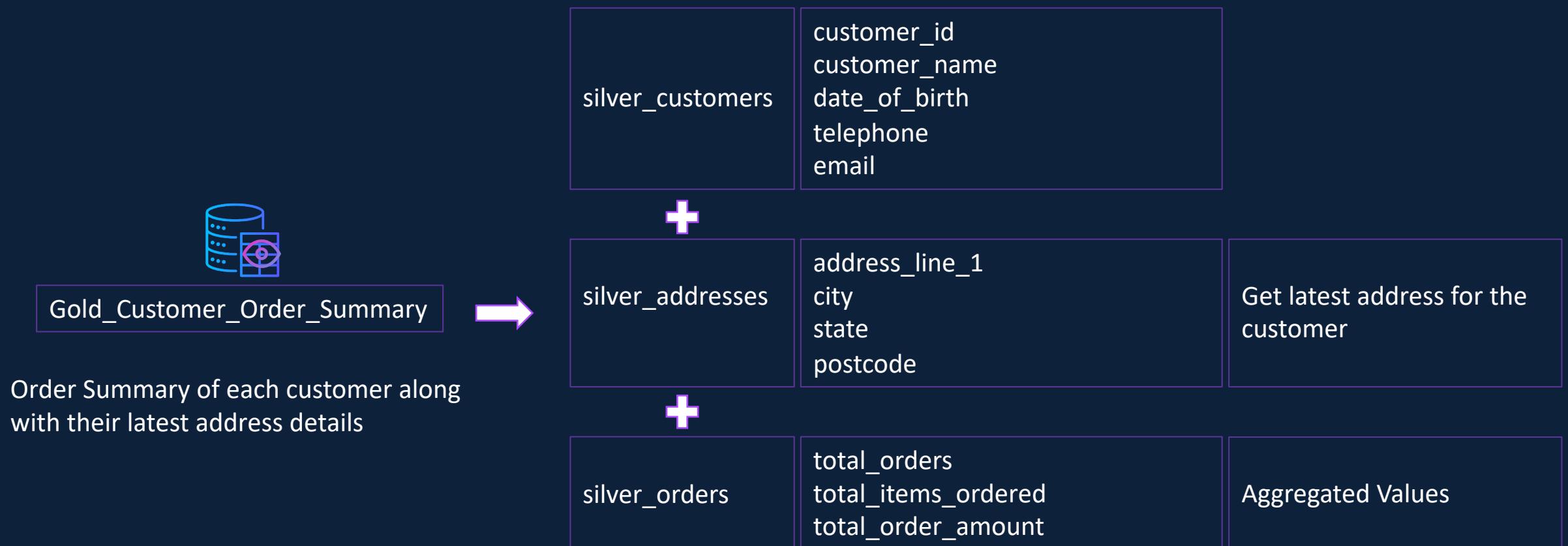
CircuitBox – Process Orders Data



Delta Live Tables (DLT) Project - CircuitBox



CircuitBox – Create Customer Order Summary



Delta Live Tables - Programming



Streaming Tables

Delta Table to which streams write data

Reads from Eventhub, Kafka, Cloud Files, Delta Tables etc.

Offers exactly once guarantees

Incremental Data Ingestion Workloads

Low latency transformations

Massive amount of data

DML operations allowed



Materialized Views

Delta Table created from the result of a query

Full Refresh Data Ingestion Workloads

Build aggregate tables for reporting

Improve latency of BI reports

Data Transformation Workloads

No DML operations allowed - Change the query instead.



Views

No physical storage of the data

Scope is limited to the pipeline

Cannot be published to Hive Metastore/Unity Catalog

Store intermediate results to reduce complexity

To enforce data quality constraints

Delta Live Tables – Legacy Terminology

New Terminology



Legacy Terminology

Materialized View

CREATE OR REFRESH MATERIALIZED VIEW

LIVE Table

CREATE OR REFRESH LIVE TABLE

Streaming Table

CREATE OR REFRESH STREAMING TABLE

Streaming LIVE Table

CREATE OR REFRESH STREAMING LIVE TABLE

Databricks SQL

Databricks SQL

Component of the Databricks Lakehouse Platform, that helps deliver data analytics and reporting on the data stored in the Data Lakehouse

Who is it for?

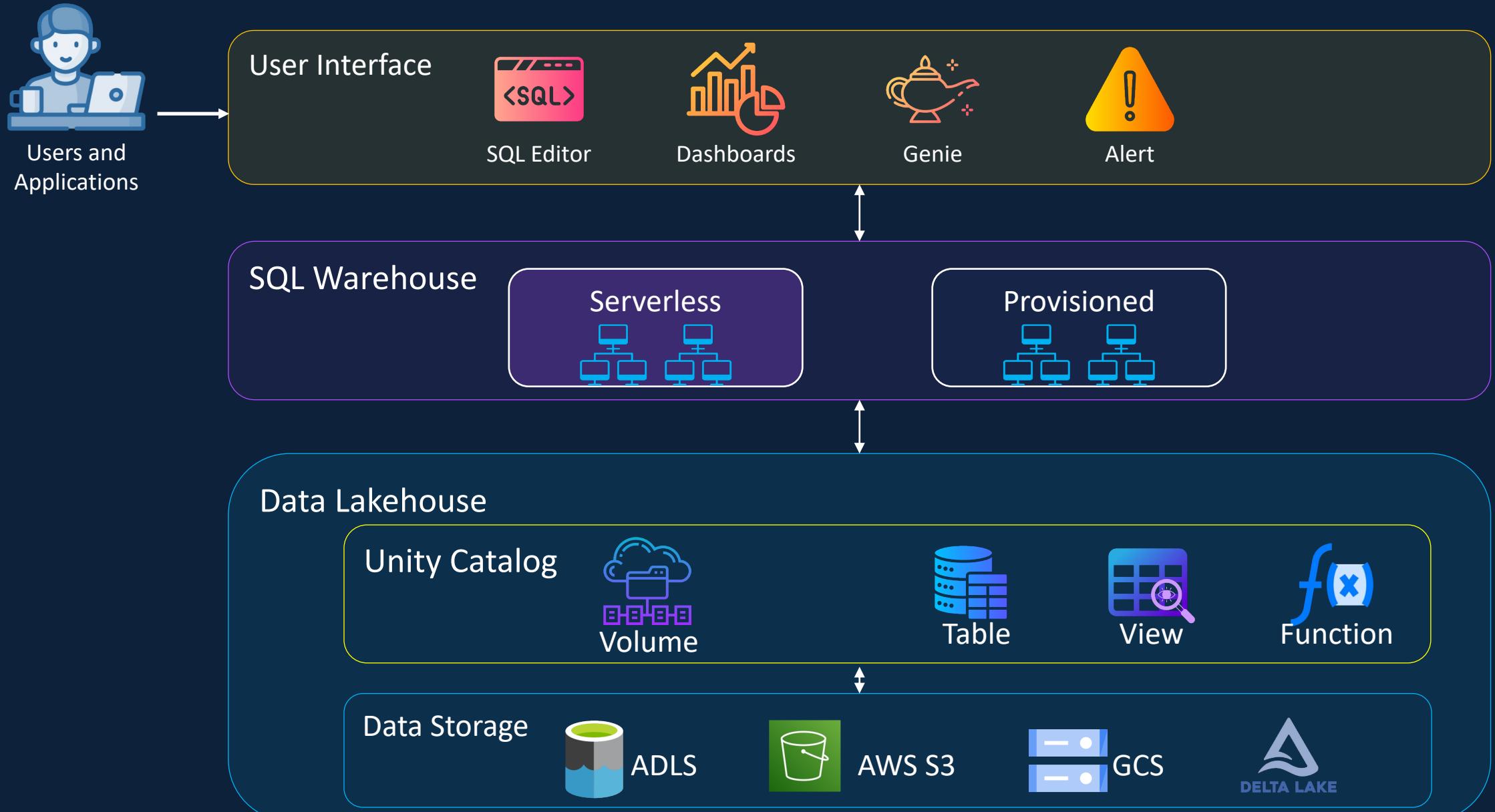
Enables Data Analysts to perform Ad-hoc Analysis, Data Visualizations and Dashboarding

Enables Business Users to extract insights from the data via Dashboards and AI/BI Genie

Helps Data Scientists to explore data quickly before writing advanced analytics in Databricks notebooks

Data Engineers should be aware of configuring clusters, alerting, scheduling, performance tuning etc.

Databricks SQL



Data Governance

Data Governance

What is Data Governance

Framework of policies, process and technologies

Ensures data accuracy, security, and compliance

Defines how data is managed, accessed, and protected

Data Governance

Why do we need Data Governance

Prevents security risks, compliance violations, and poor decisions

Avoids data inconsistencies and unauthorized access

Improves operational efficiency and risk management

Data Governance

Key Benefits of Data Governance

Ensures data quality

Strengthens security

Ensures compliance

Optimizes decision-making

Reduces costs & risks

Data Governance

Data Governance in Databricks

Unity Catalog

Delta Lake

Delta Sharing

Cluster & Workspace Logging

Unity Catalog – Data Governance Solution



Unity Catalog – Data Governance Solution



Unity Catalog

Unity Catalog is a Databricks offered unified solution for implementing data governance in the Data Lakehouse

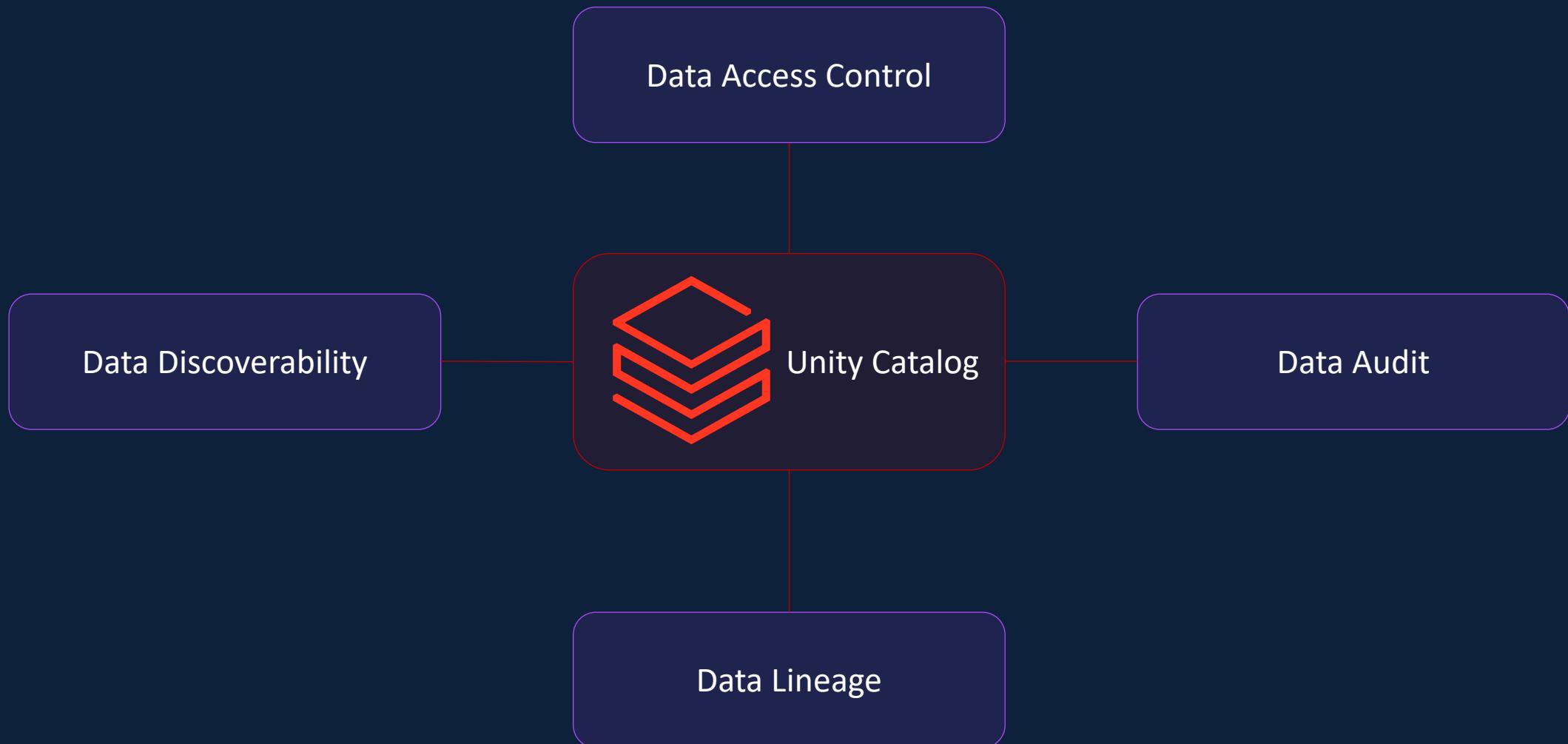
Unity Catalog – Data Governance Solution



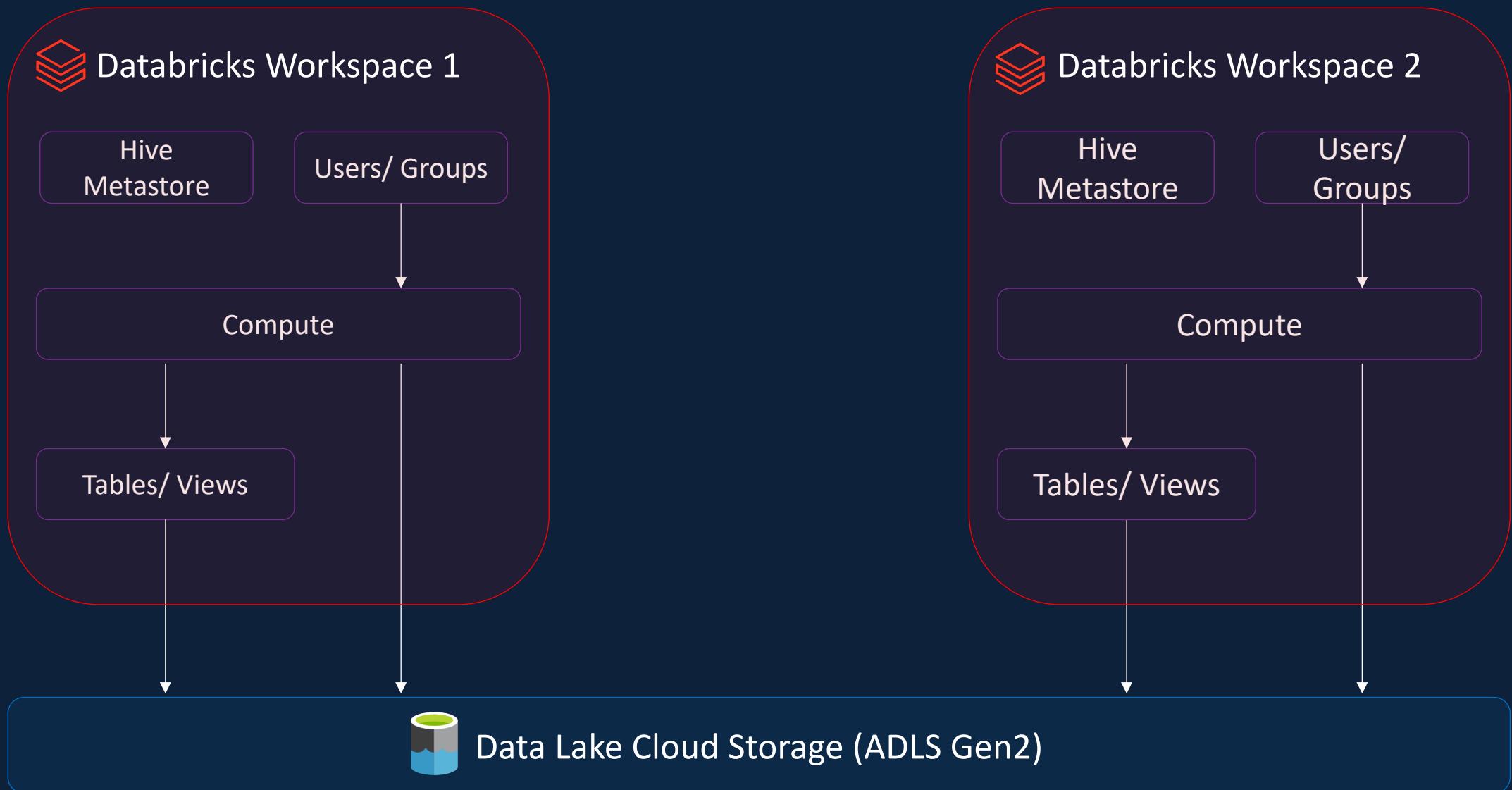
Unity Catalog

Unity Catalog is a Databricks offered **unified solution** for implementing **data governance** in the Data Lakehouse

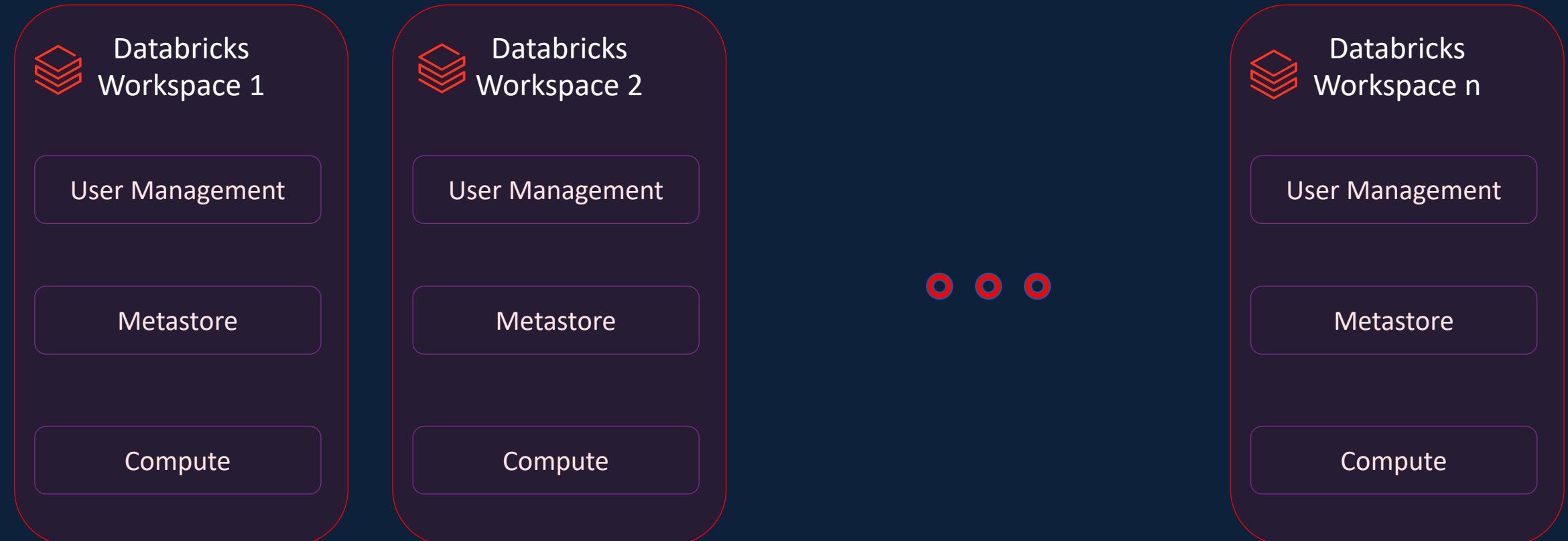
Unity Catalog – Data Governance Solution



Unity Catalog – Data Governance Solution



Unity Catalog – Data Governance Solution



Unity Catalog – Data Governance Solution

Without Unity Catalog

Databricks
Workspace 1

User Management

Metastore

Compute

Databricks
Workspace 2

User Management

Metastore

Compute

Databricks Account

With Unity Catalog

Databricks Unity Catalog

User Management

Metastore

Databricks
Workspace 1

Compute

Databricks
Workspace 2

Compute

Databricks Account

Unity Catalog – Data Governance Solution

Without Unity Catalog

Databricks
Workspace 1

User Management

Metastore

Compute

Databricks
Workspace 2

User Management

Metastore

Compute

Databricks Account

With Unity Catalog

Databricks Unity Catalog

User Management

Metastore

Databricks
Workspace 1

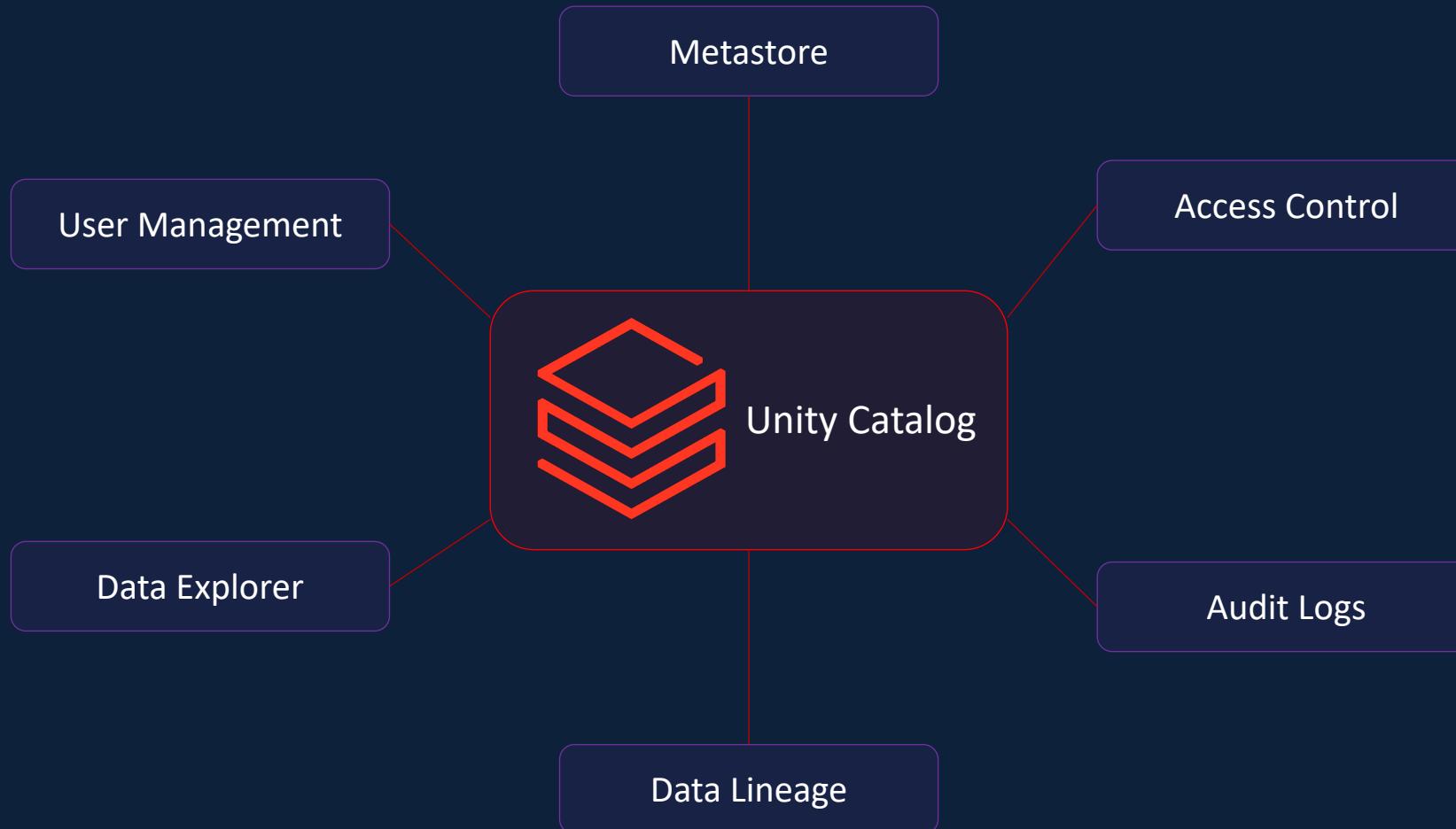
Compute

Databricks
Workspace 2

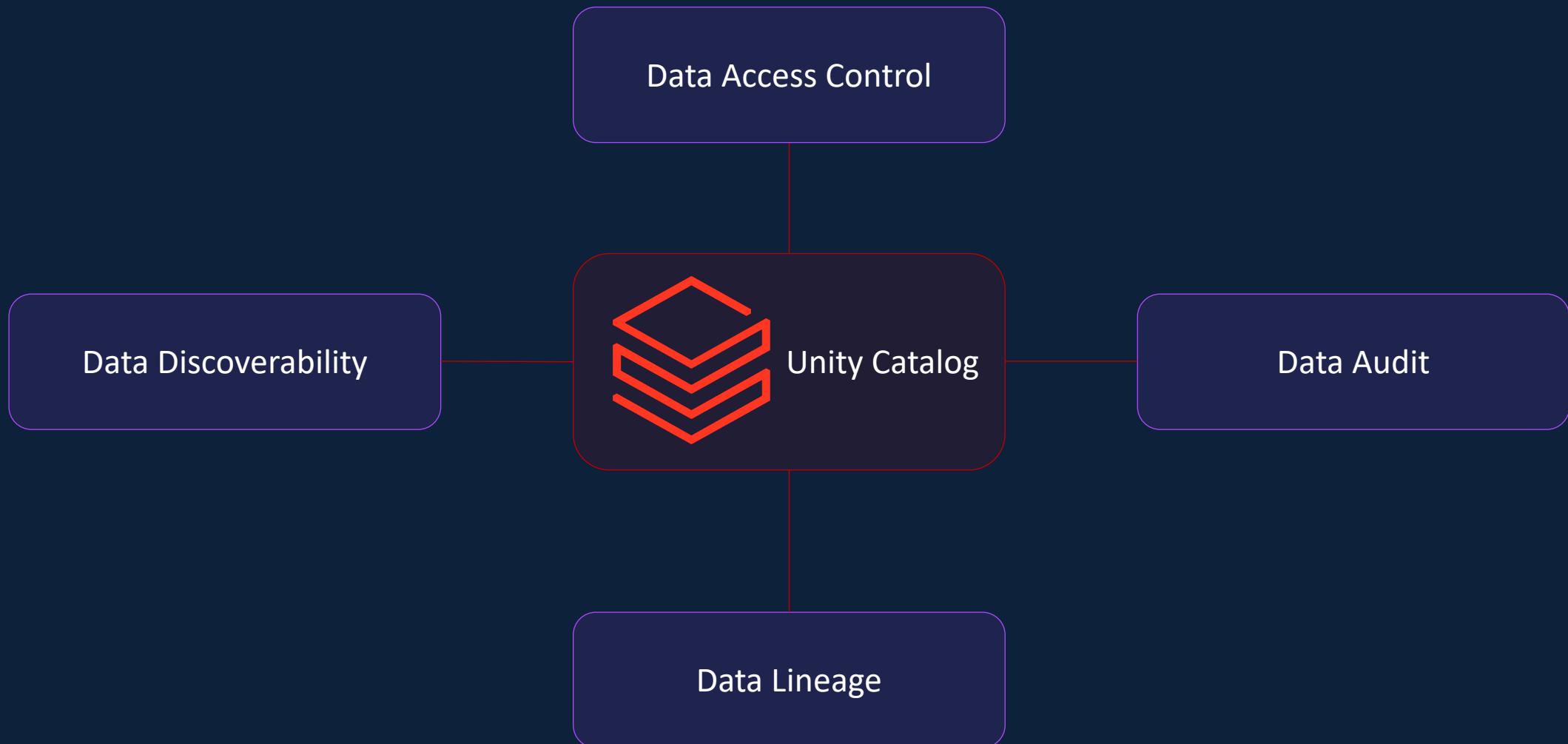
Compute

Databricks Account

Unity Catalog – Data Governance Solution



Unity Catalog – Data Governance Solution



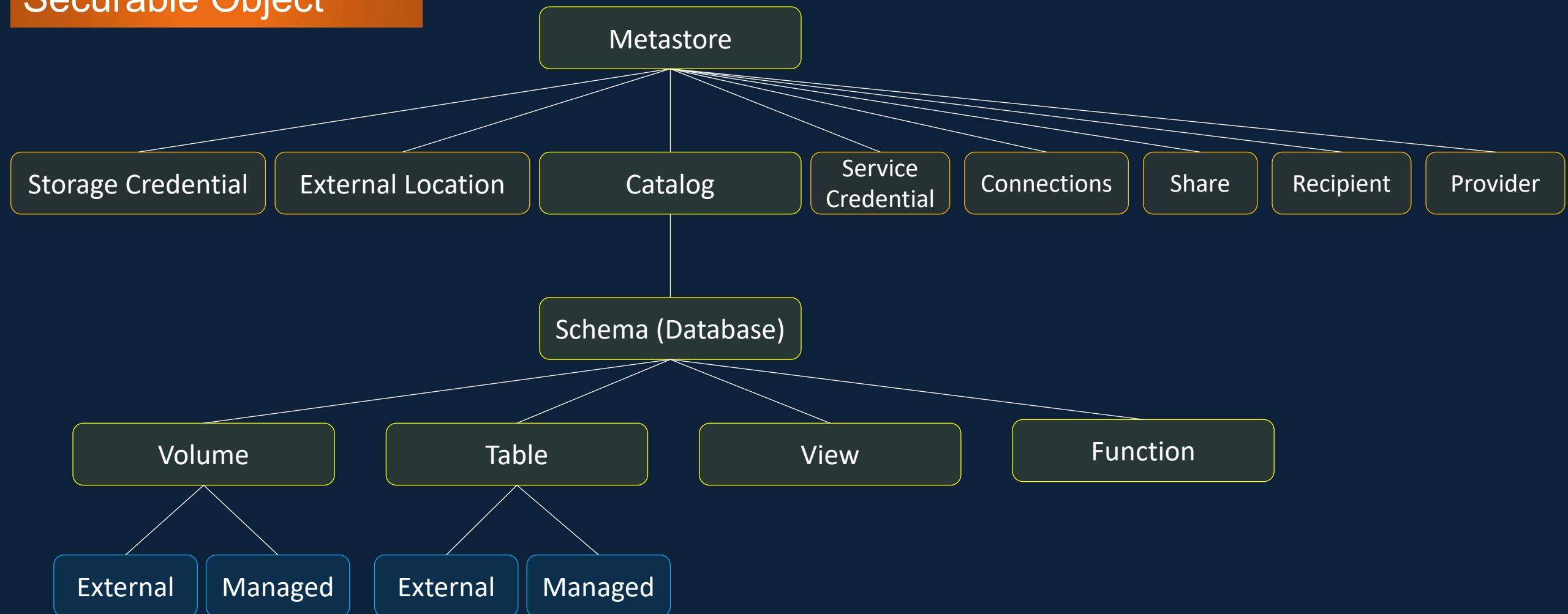
Data Governance - Demo

Unity Catalog - Security Model



Unity Catalog - Security Model

Securable Object

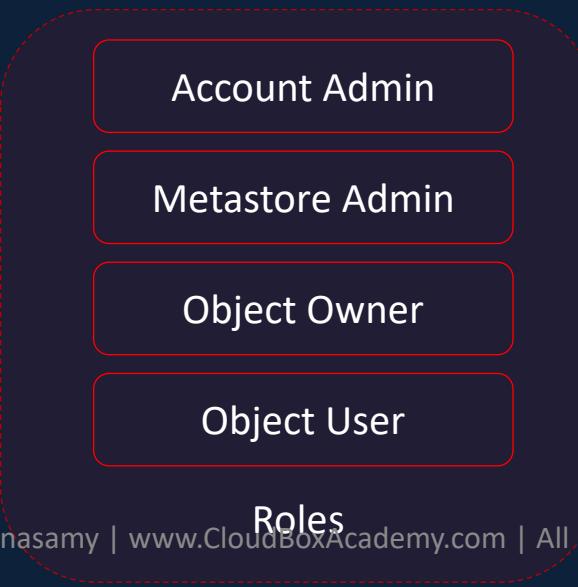
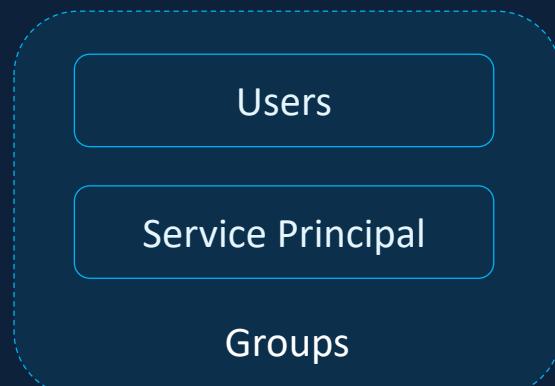


Unity Catalog - Security Model



Unity Catalog - Security Model

Role Based Access



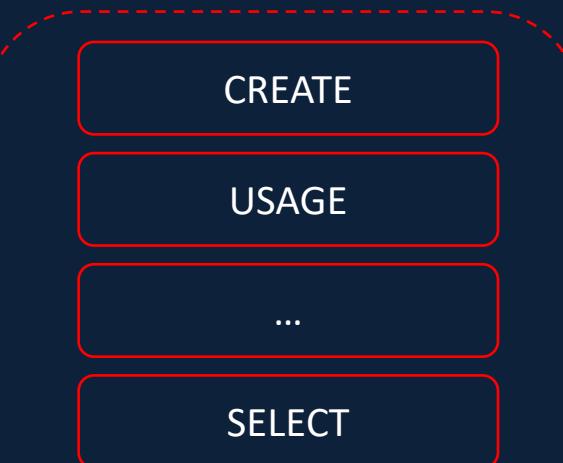
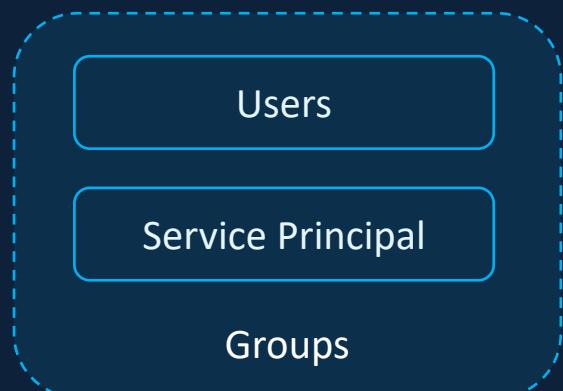
Unity Catalog - Security Model

Access Control List



GRANT <privilege> ON <securable object> TO <identity>

REVOKE <privilege> ON <securable object> FROM <identity>

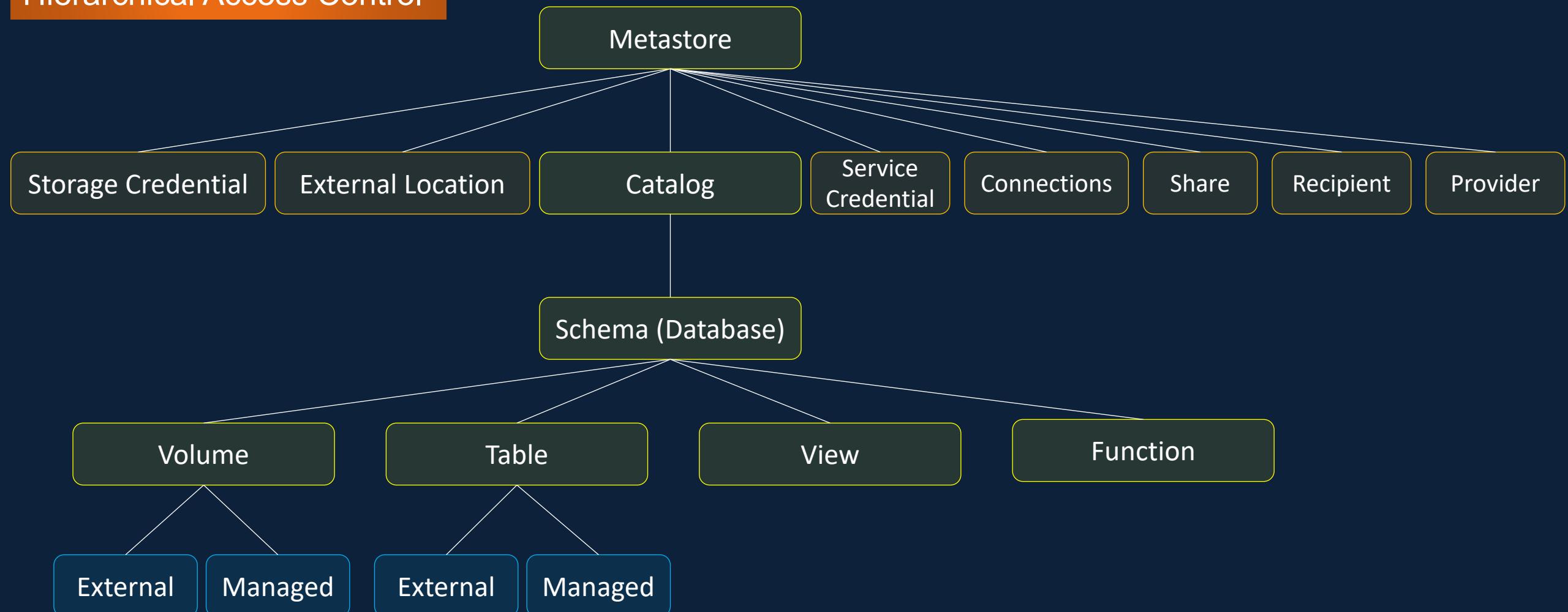


GRANT SELECT ON TABLE customers TO sales

REVOKE SELECT ON TABLE customers FROM sales

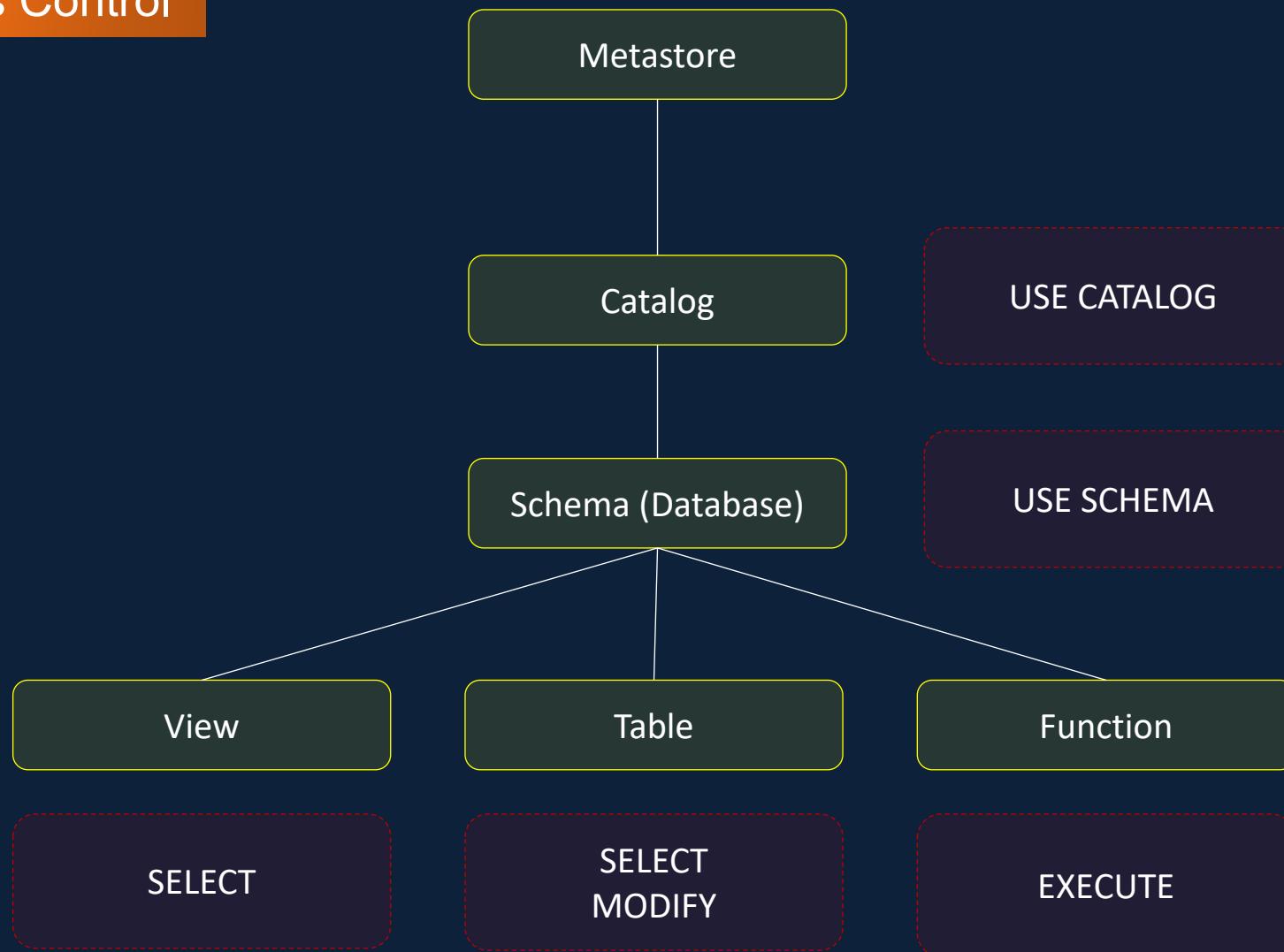
Unity Catalog - Security Model

Hierarchical Access Control



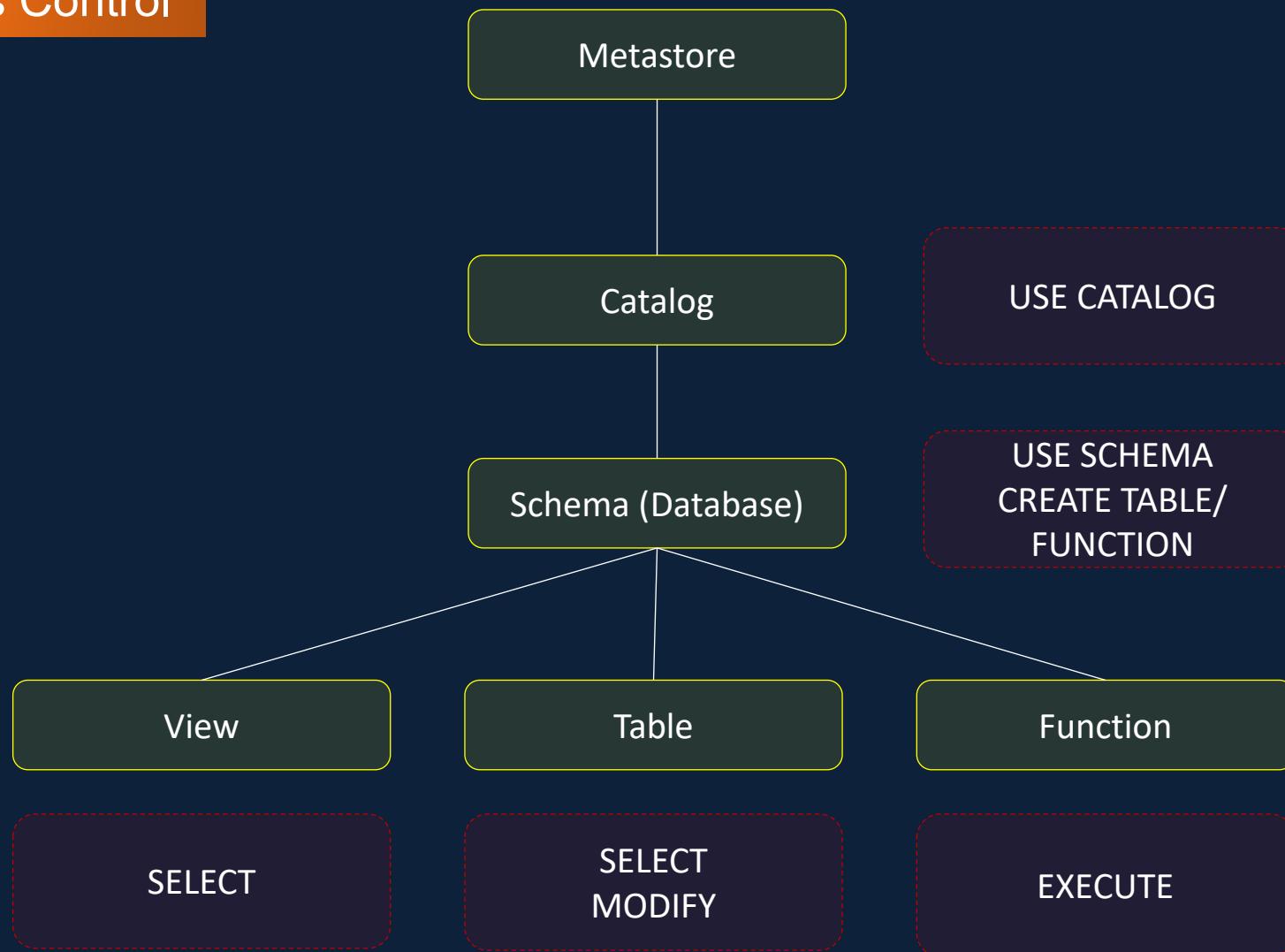
Unity Catalog - Security Model

Hierarchical Access Control



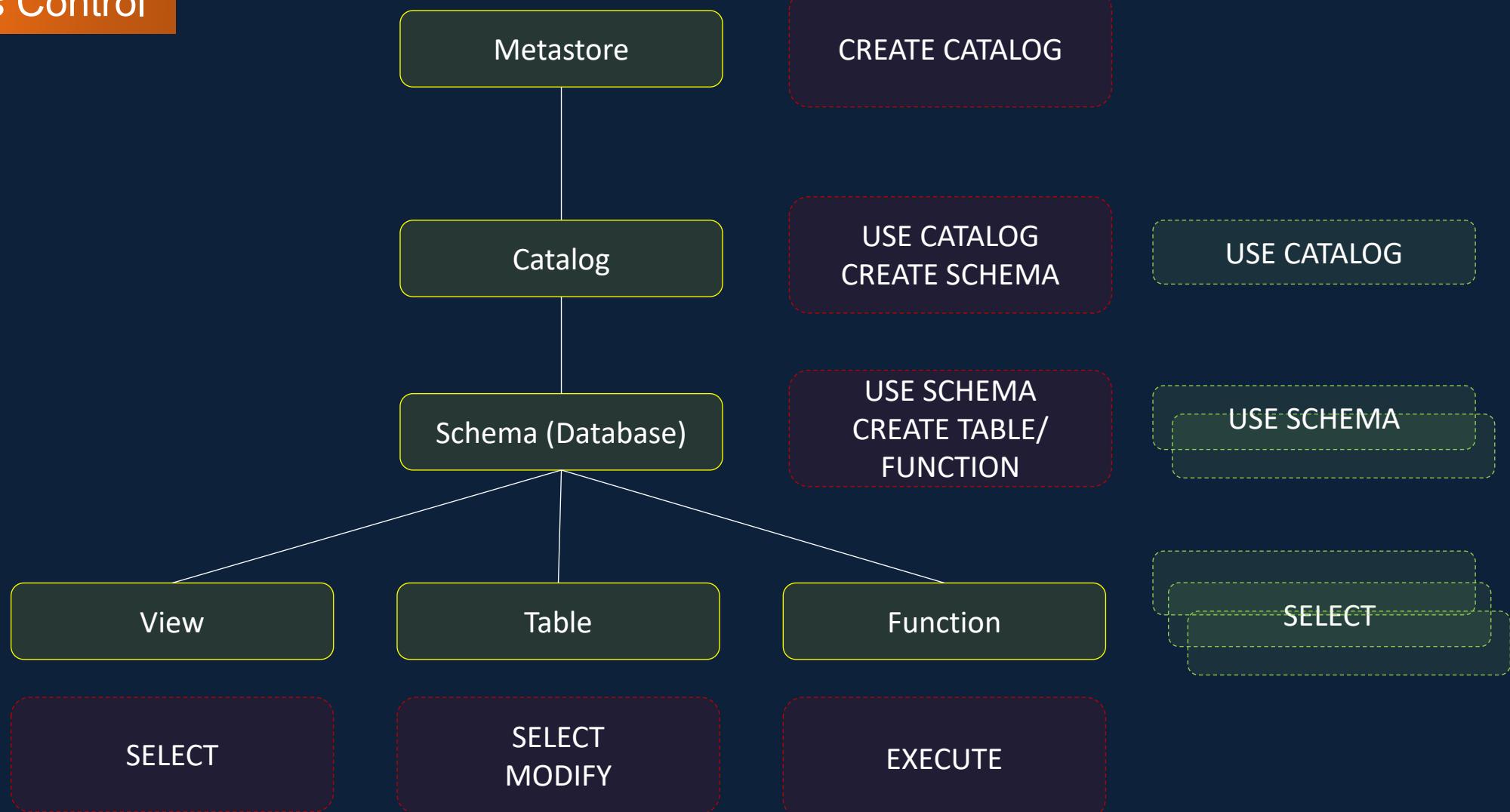
Unity Catalog - Security Model

Hierarchical Access Control



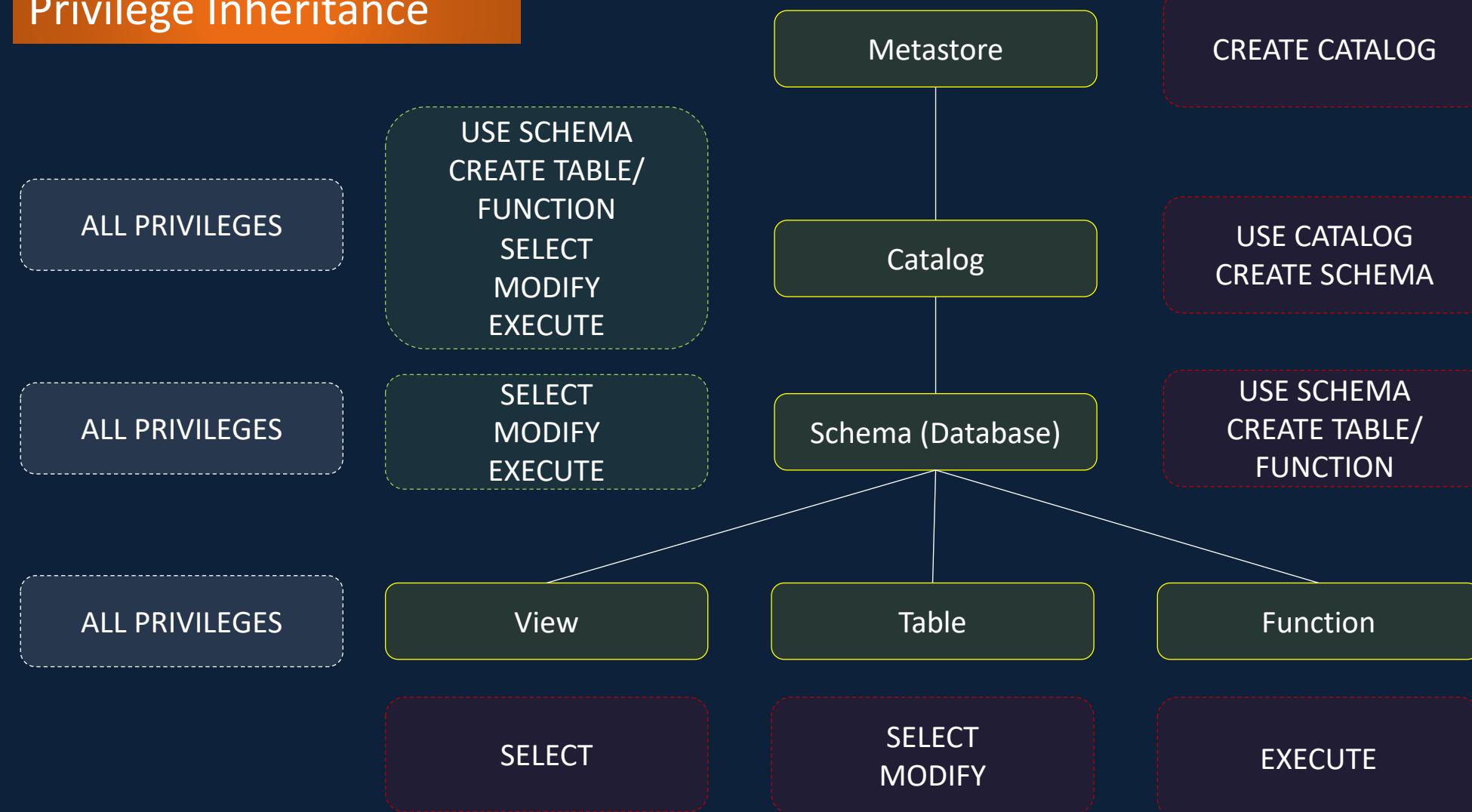
Unity Catalog - Security Model

Hierarchical Access Control



Unity Catalog - Security Model

Privilege Inheritance



Delta Sharing

Delta Sharing

Delta Sharing is an open data sharing protocol that enables secure sharing of data across business units, customers, suppliers and partners.

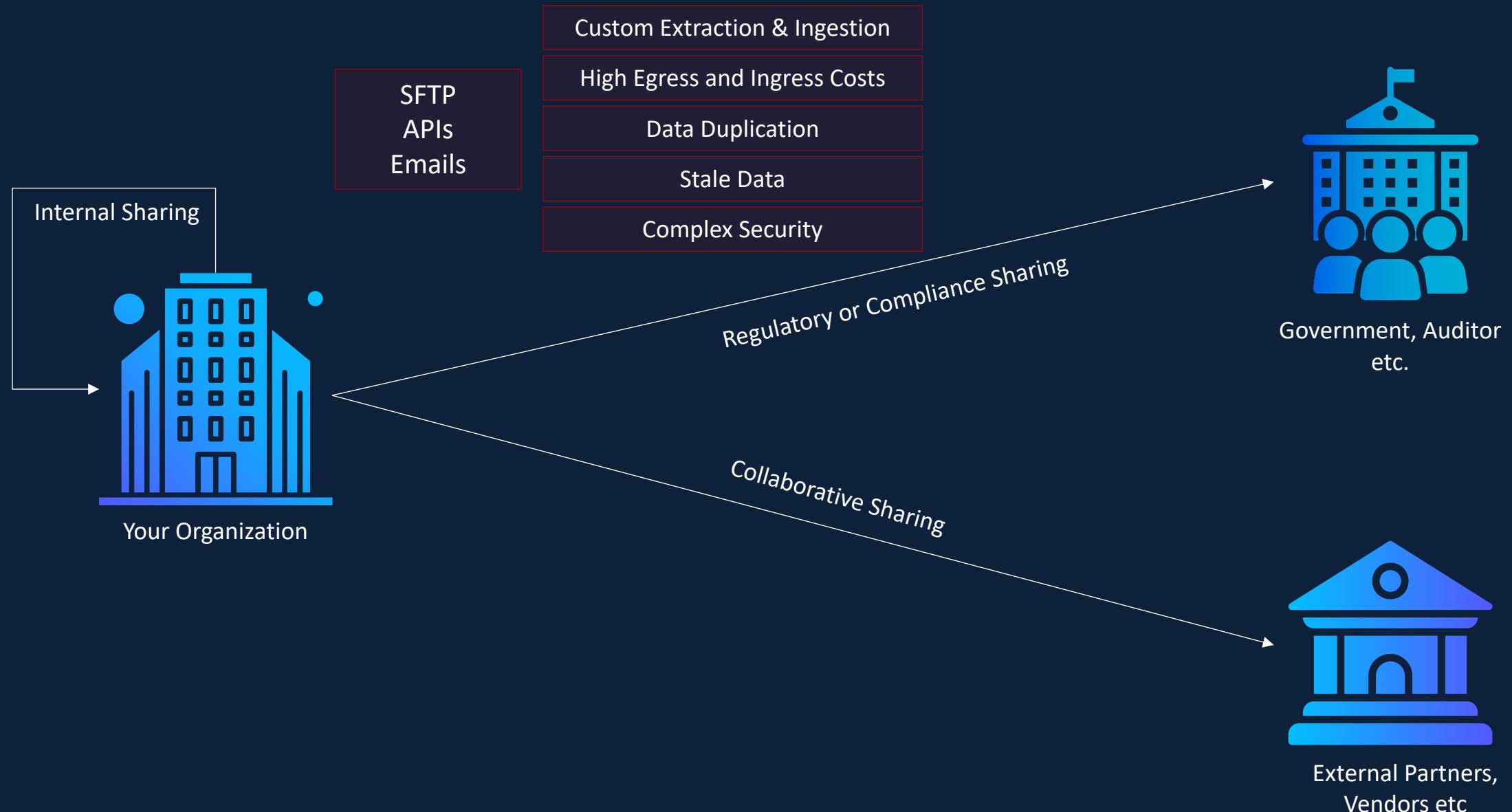
Developed by Databricks, contributed to open-source Delta Lake project & governed under Linux Foundation

Available as Fully managed inside the Databricks platform in workspaces enabled with Unity Catalog.

Native integration with Databricks Platform allows sharing of Notebooks, Dashboards, AI Models, Volumes etc.

Native integration with Unity Catalog for Governance and Security

Data Sharing

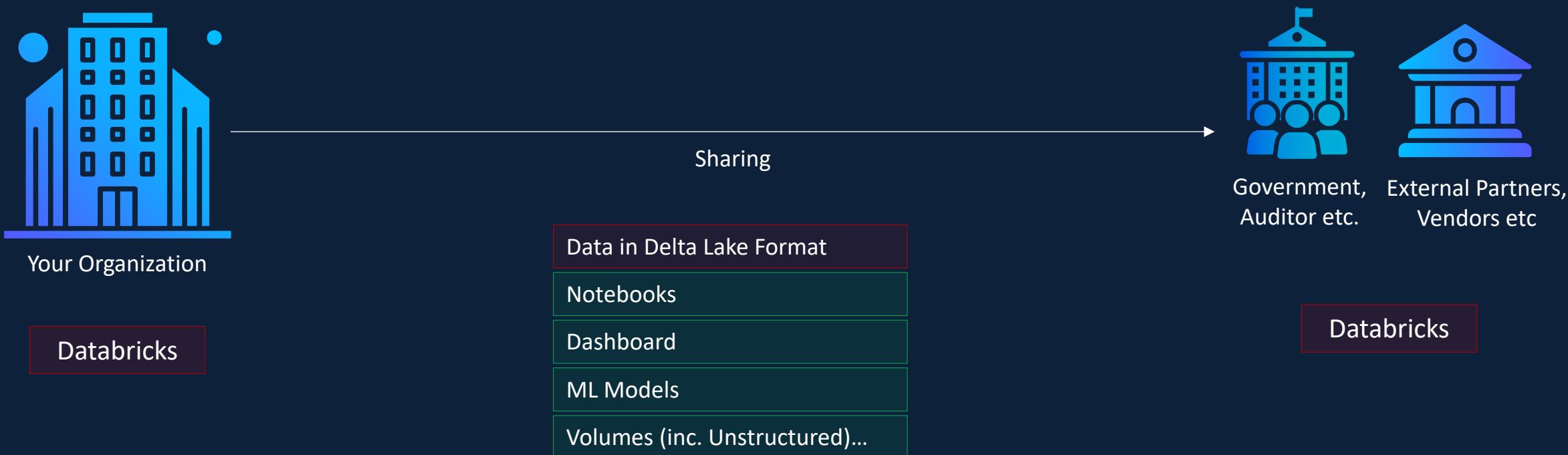


Delta Sharing

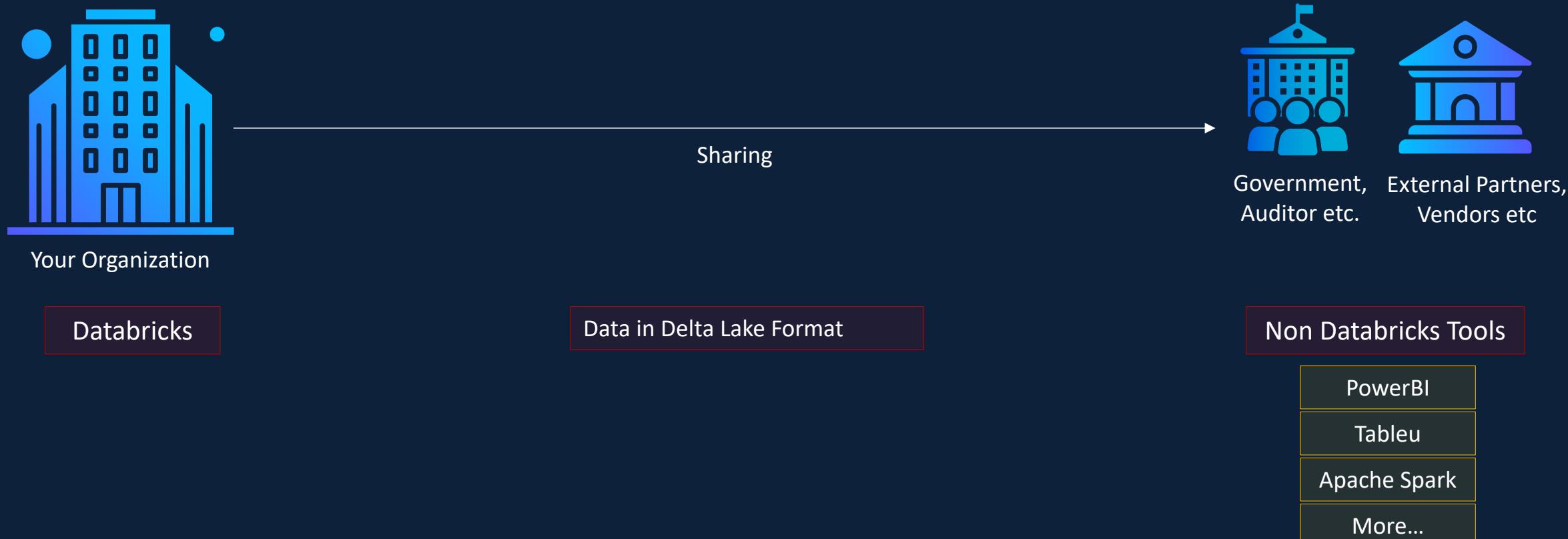


Delta Sharing Protocols

1. Databricks to Databricks Sharing Protocol



Delta Sharing Protocols



Delta Sharing Protocols



Your Organization

Non Databricks Tools

PowerBI
Tableau
Apache Spark
More...

Sharing

1. Databricks to Databricks Sharing Protocol
2. Databricks Open Sharing Protocol
3. Open-source Delta Sharing Server – Customer Managed



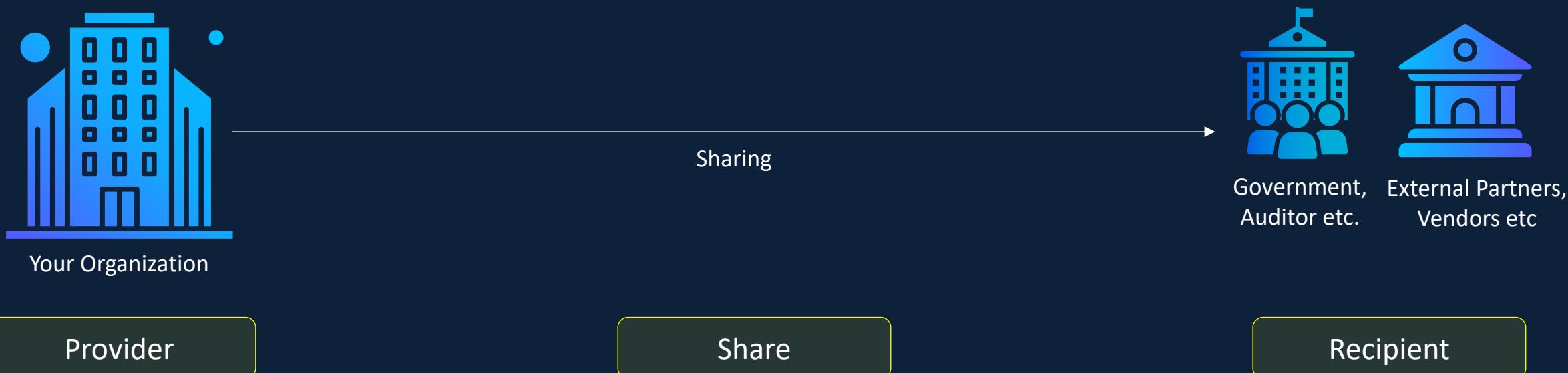
Data in Delta Lake Format

Non Databricks Tools

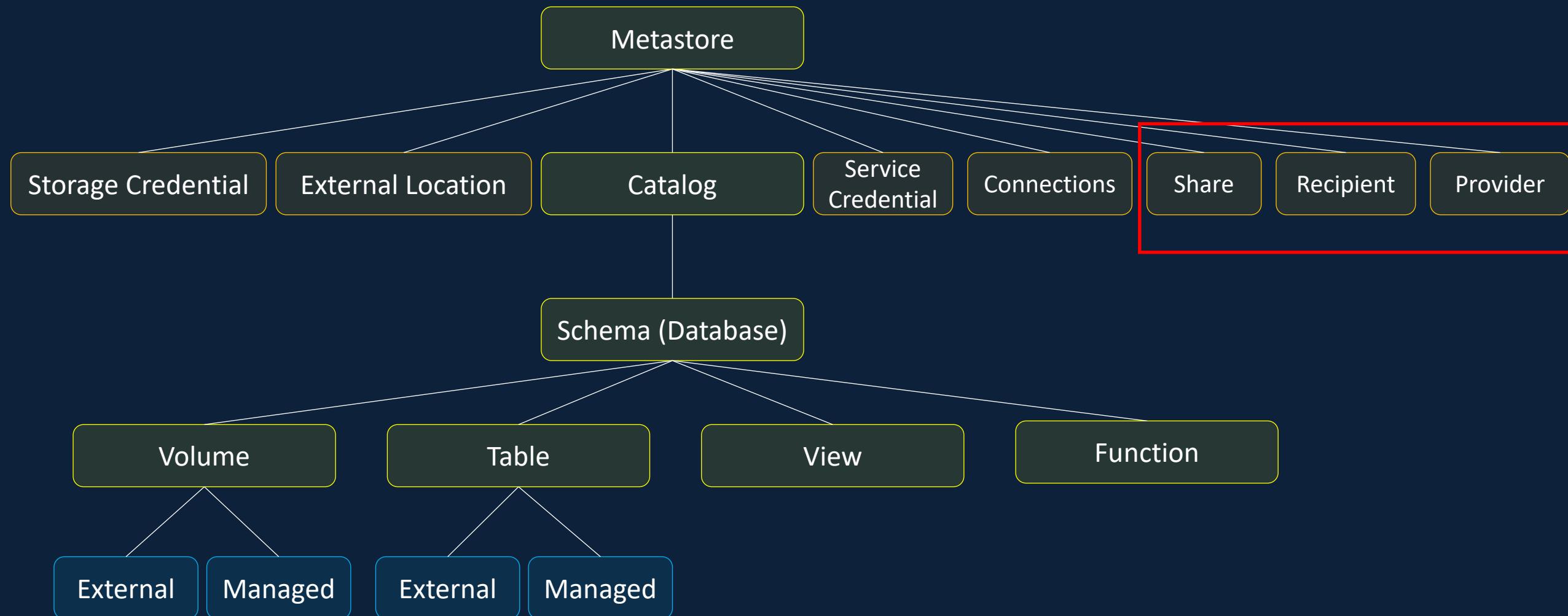
PowerBI
Tableau
Apache Spark
More...

Delta Sharing - Key Concepts

1. Databricks to Databricks Sharing Protocol
2. Databricks Open Sharing Protocol
3. Open-source Delta Sharing Server – Customer Managed

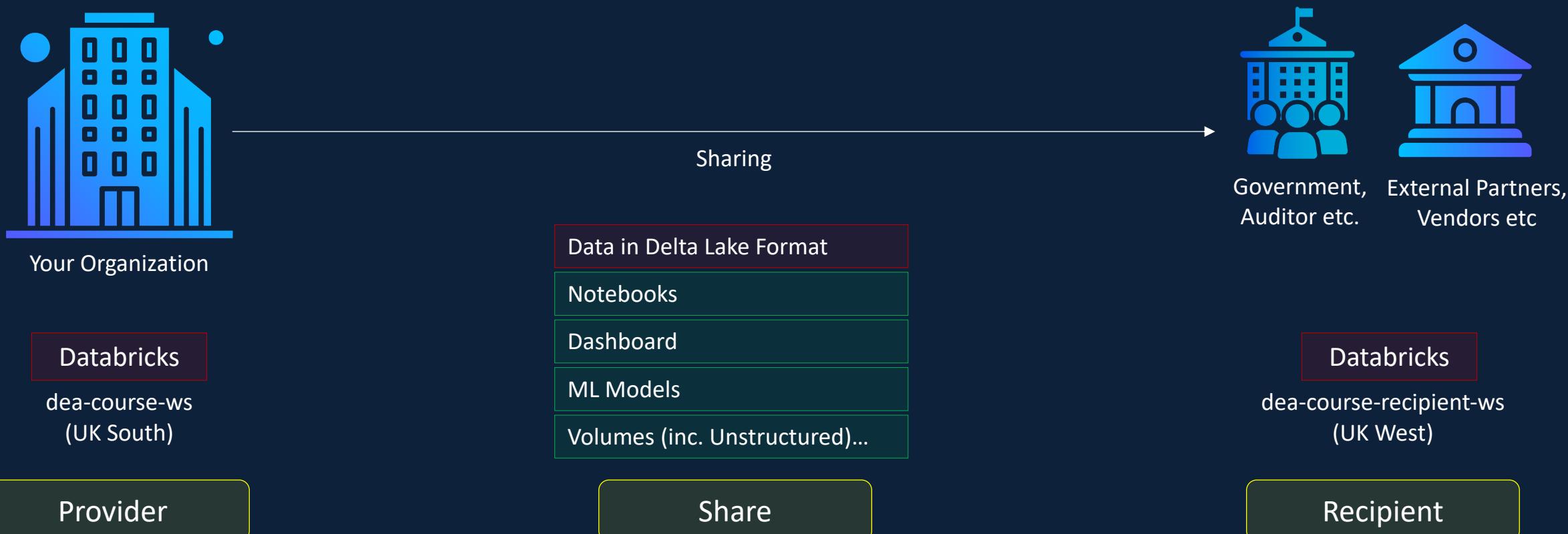


Unity Catalog Object Model



Databricks to Databricks Delta Sharing

1. Databricks to Databricks Sharing Protocol



Databricks Open Sharing Protocol



Your Organization

Databricks

Sharing



Government, External Partners,
Auditor etc. Vendors etc

Data in Delta Lake Format

Non Databricks Tools

PowerBI

Tableau

Apache Spark

More...

Provider

Share

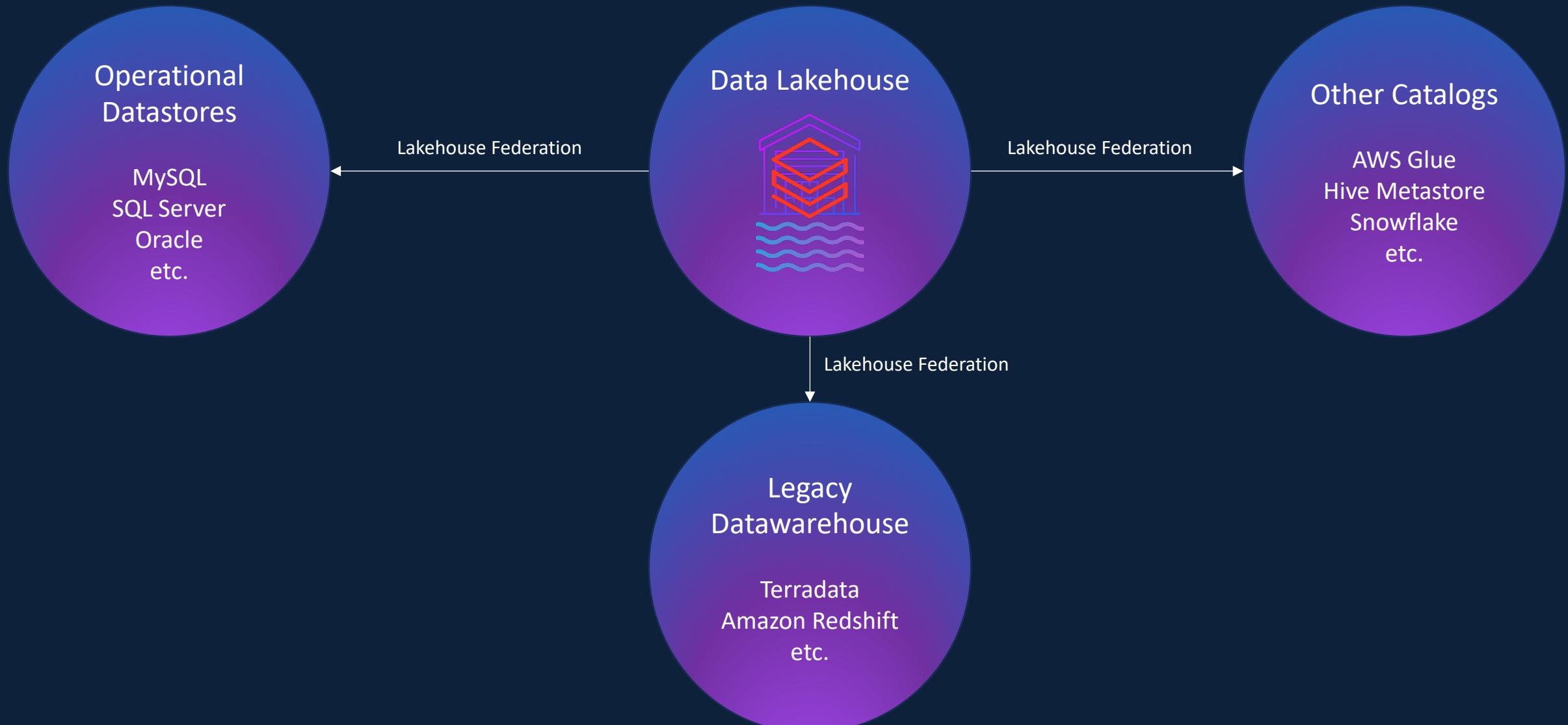
Recipient

Lakehouse Federation

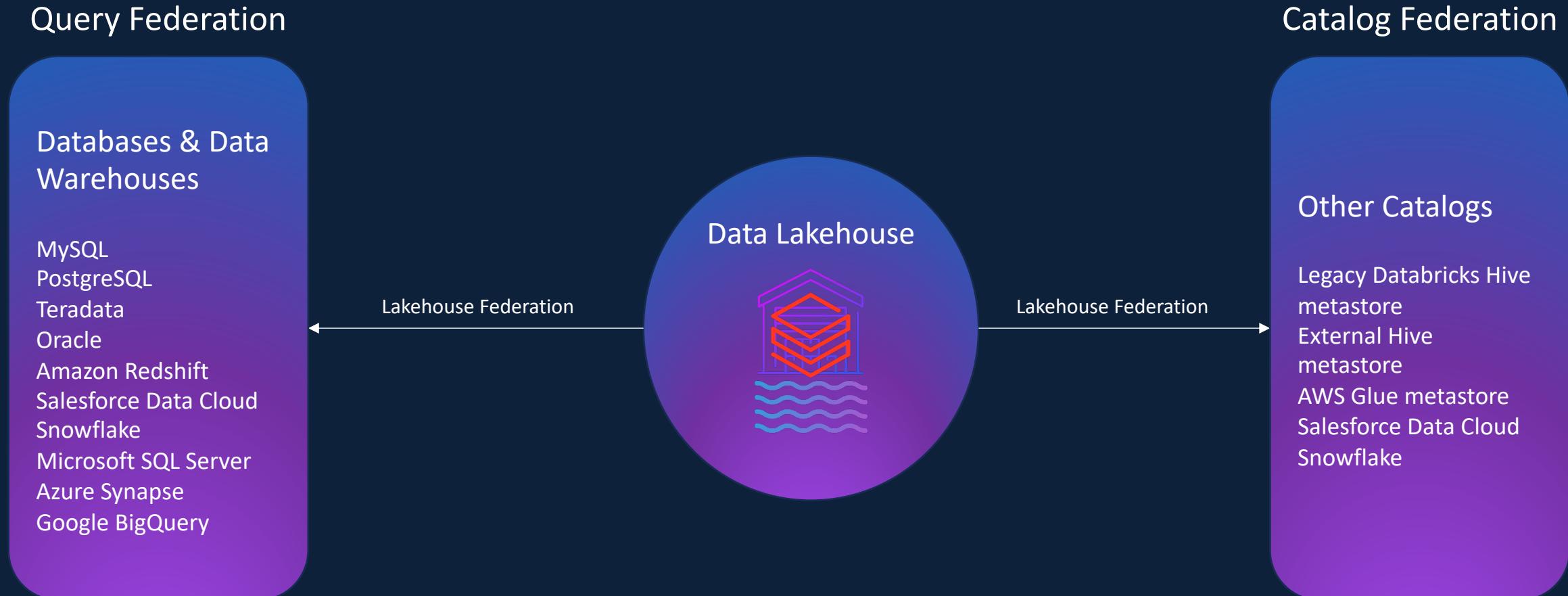
Lakehouse Federation

Lakehouse Federation is a capability in Databricks that allows you to access and govern data stored outside of the Lakehouse without physically moving or copying it into the Lakehouse.

Lakehouse Federation



Lakehouse Federation



Lakehouse Federation Types

Query Federation

Direct JDBC connection to external databases (eg. MySQL, SQL Server, Redshift)

Query executed on the external database

Performance/cost depend on source system

Can join with Lakehouse data

Best for ad-hoc or PoC

Catalog Federation

Connects to external catalogs (AWS Glue, Hive Metastore, Snowflake)

Query executed on Databricks compute

More cost-effective and performance-optimized

Can join with Lakehouse data

Best for migration or hybrid models

Lakehouse Federation Implementation

Query Federation

Unity Catalog enabled

Create connection (JDBC + credentials)

Create foreign catalog

Grant privileges

Run queries → pushed down to external DB

Catalog Federation

Unity Catalog enabled

Create connection to external catalog

Create storage credential + external location

Create foreign catalog

Grant privileges

Run queries → executed on object storage

Lakehouse Federation Implementation

Query Federation

Unity Catalog enabled

Create connection (JDBC + credentials)

Create foreign catalog

Grant privileges

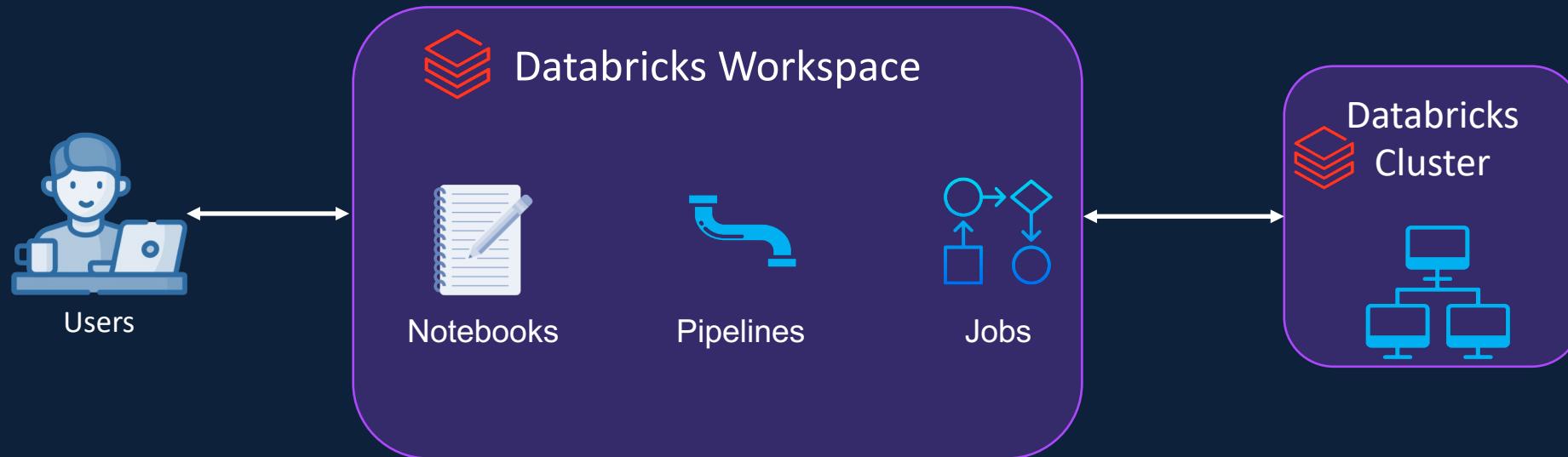
Run queries → pushed down to external DB

Catalog Explorer UI

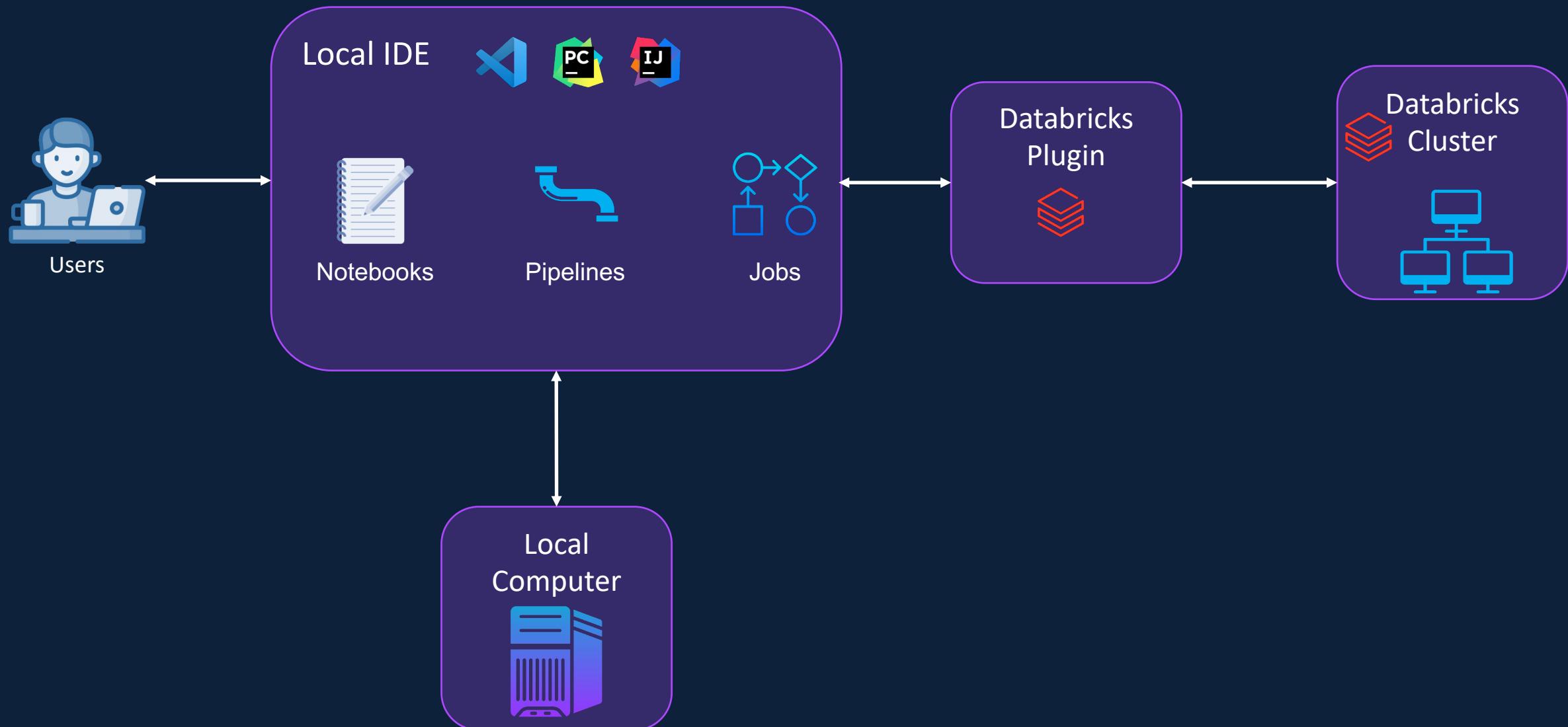
SQL Statements

Databricks Connect

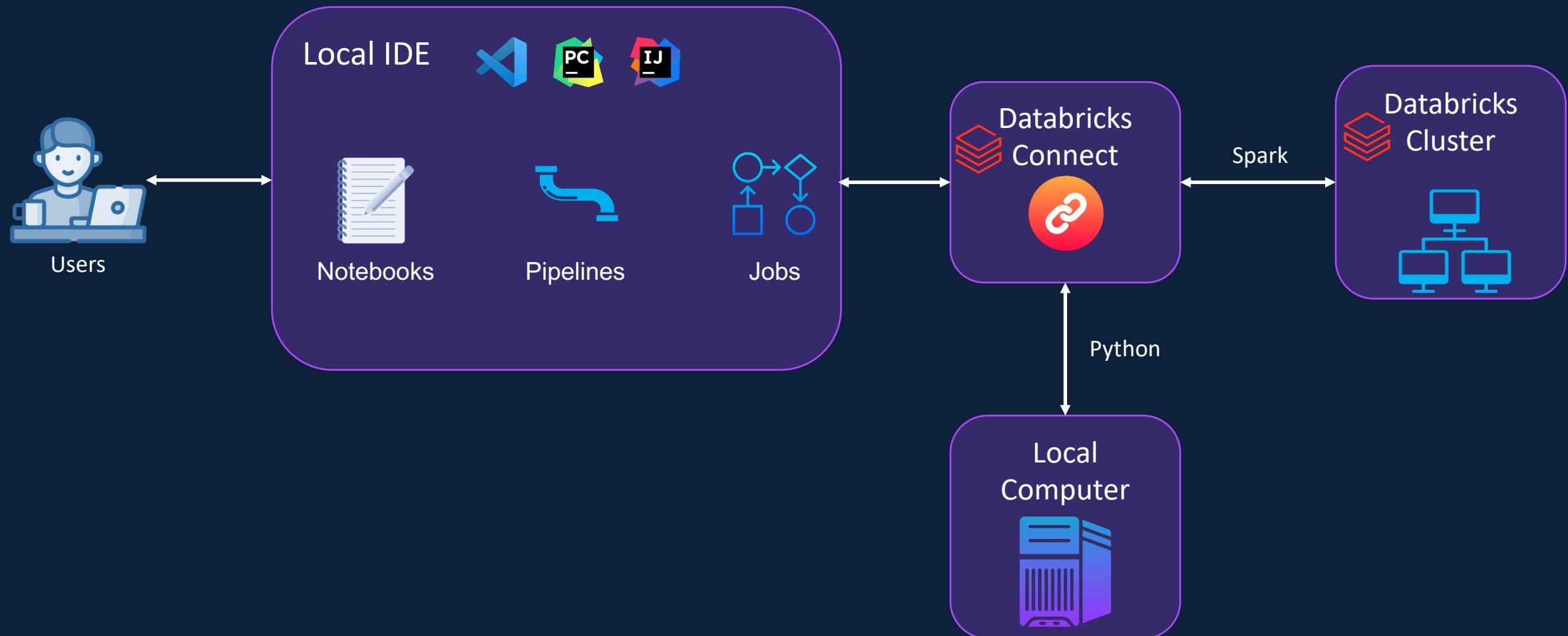
Local Development



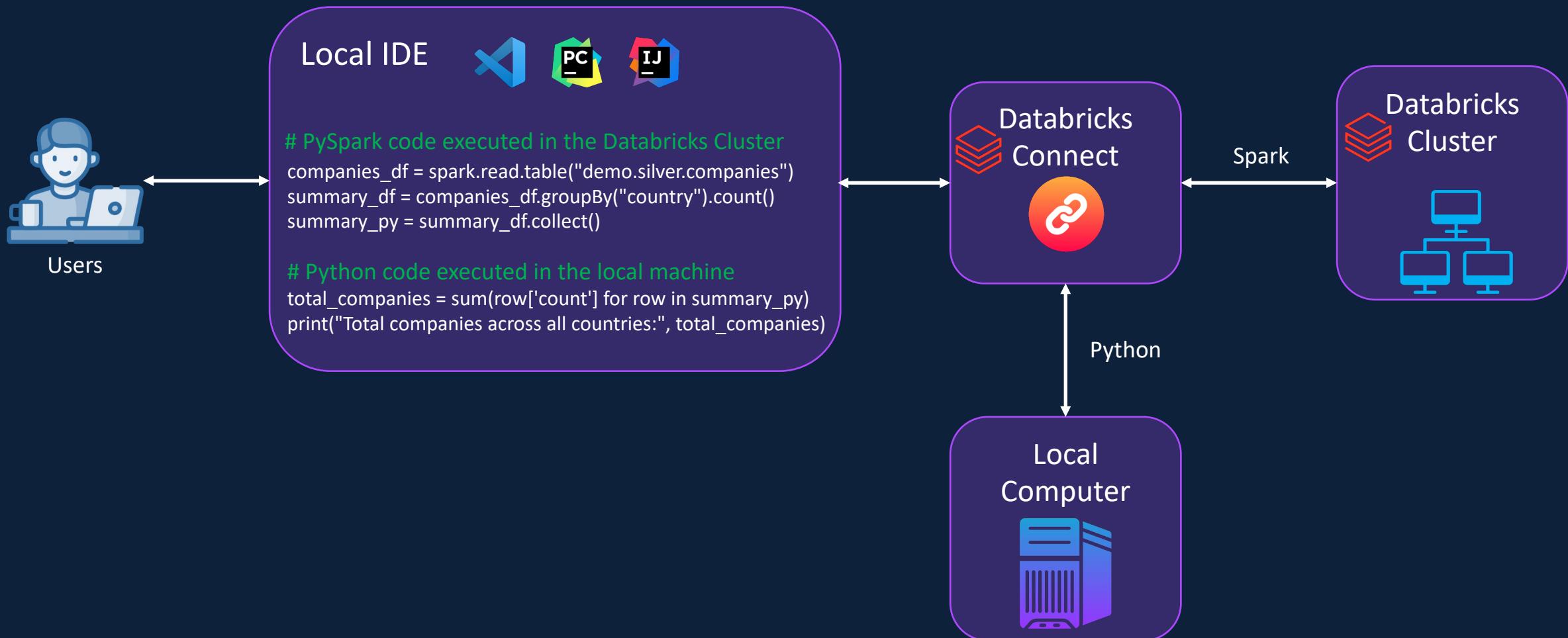
Local Development



Local Development



Local Development



Local Development Environment Set-up

Install Python

Install Visual Studio Code

Install Databricks Plugin

Create Databricks Project

Configure Access to Databricks

Run Code on Databricks Cluster

Databricks Connect Set-up

Create Python Virtual Environment

Configure Databricks Connect

Debug/ Run Code via Databricks Connect

Databricks Asset Bundles (DABs)

Databricks Asset Bundles (DABs)

Databricks Asset Bundles are a tool to facilitate the adoption of software engineering best practices, including source control, code review, testing, and continuous integration and delivery (CI/CD), for your data and AI projects.

- Databricks

Databricks Project



Modern Deployment Approach



Databricks Asset Bundles



Legacy Deployment Approach



Databricks REST APIs/ CLI

Terraform Provider

Databricks Labs - DBX

Databricks Asset Bundles

Collection of:

Code (notebooks, Python, SQL)

Configurations for Databricks resources (jobs, pipelines, clusters, MLflow, etc.)

Environment settings (dev, test, prod)

Represented in **YAML**

Deployed through **CI/CD tools** like GitHub Actions, Azure DevOps

Provides a standard, repeatable way to deliver Databricks projects

Structure of a Bundle

Structure of a Bundle

Defined in `databricks.yml`

Entry point of a Databricks Asset Bundle (DAB) Project

Uses YAML (Declarative Format, Human Readable)

Structure of a Bundle

bundle → project name & metadata

resources → jobs, pipelines, clusters, MLflow, etc.

targets → environment specific settings (dev, test, prod)

Structure of a Bundle

```
# Databricks Asset Bundle for dab_demo_project
bundle:
  name: dab_demo_project

resources:
  jobs:
    dab_demo_project_job:
      name: dab_demo_project_job
      tasks:
        - task_key: dab_demo_notebook
          notebook_task:
            notebook_path: src/dab_demo_notebook.ipynb
            job_cluster_key: job_cluster
      job_clusters:
        - job_cluster_key: job_cluster
          new_cluster:
            spark_version: 15.4.x-scala2.12
            node_type_id: Standard_D3_v2
            data_security_mode: SINGLE_USER
            num_workers: 1

targets:
  dev:
    default: true
    mode: development
    workspace:
      host: https://adb-12345.14.azuredatabricks.net
  prod:
    workspace:
      host: https://adb-67890.14.azuredatabricks.net
```

Structure of a Bundle

variables → define reusable values (like parameters)

include → pull in configs from other files (for modular bundles)

run_as → specify the identity (user or service principal) to run jobs

artifacts → reference libraries, wheels, or other external assets

sync → control what gets synced between local and workspace



Exam Overview

Databricks Certified Data Engineer Associate

Exam Overview

Registration



Register via: <https://webassessor.com/databricks>

Registration Fee: \$200

Understand Computer/ Room Requirements

Provide one form of Identity document (e.g. Passport)

Delivery Method: Online Proctored

Test Aides Allowed: None

Exam Overview

Exam Format



Total Number of Questions: 45 [+5 Experimental]

Pass Percentage: 70% [32 Questions]

Question Type: Multiple Choice

Time Limit: 90 minutes

Validity Period: 2 Years

Exam Overview

Exam Topics



Databricks Lakehouse Platform – 24% [11 Questions]

ETL With Apache Spark – 29% [13 Questions]

Incremental Data Processing – 22% [10 Questions]

Production Pipelines – 16% [7 Questions]

Data Governance – 9% [4 Questions]



Official Exam Guide - <https://www.databricks.com/learn/certification/data-engineer-associate>

Course Disclaimer

This course is **not affiliated with or endorsed by Databricks, Inc.**

Databricks® is a registered trademark of Databricks, Inc.

This course is independently created to help you prepare for the **Databricks Certified Data Engineer Associate** exam

It's based on publicly available resources and my own experience with the platform.

Practice Exam Disclaimer

All the questions in this course are **original**.

They are designed to mirror the format and style of the actual certification exam.

We do **not** use or share live exam questions.

It's built based on publicly available resources.

Use of Practice Exam

Get used to the structure of the exam

Test your understanding of the topics

Build Confidence before taking the actual exam

Not to replace the official Databricks Exam

Congratulations!
&
Thank you





Good Luck!



Ratings & Review



Thank you