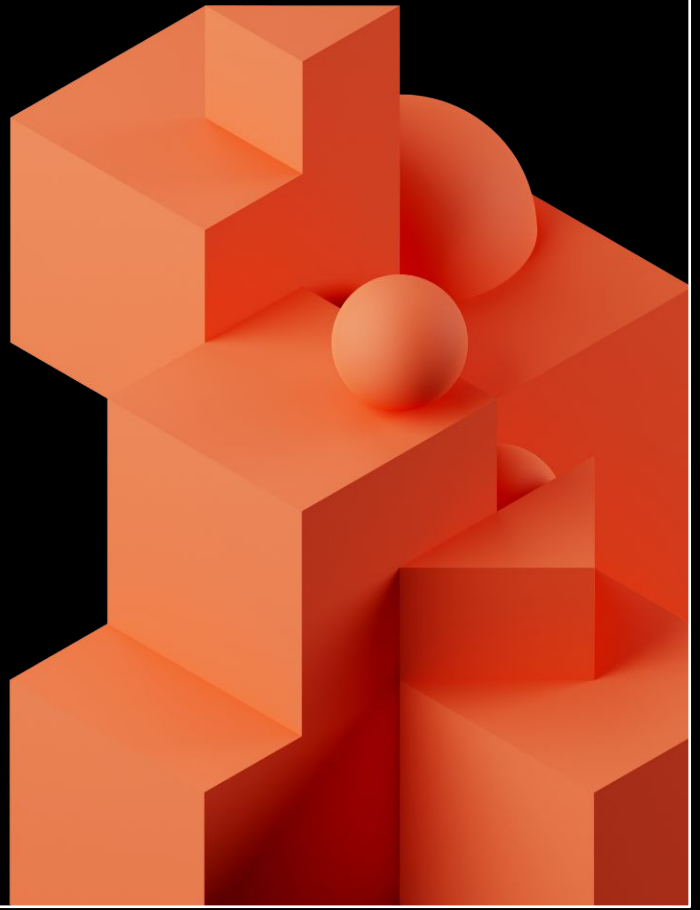
DATA+AI SUMMIT
BY databricks

Data Engineering with Databricks



Databricks Academy
2023

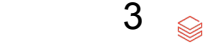
Meet your instructor

Add instructor name, Add instructor title

- Team: <add>
- Time at Databricks: <add>
- Fun fact: <Add>

Meet your classmates

- Where is everyone joining us from today (city, country)?



Meet your classmates

- How long have you been working with Databricks?

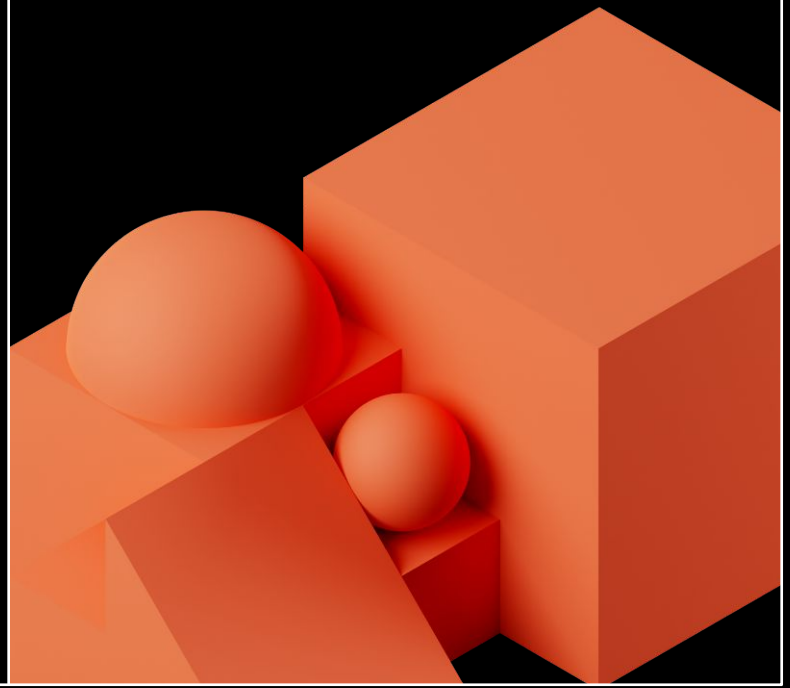
Meet your classmates

- What has your experience working with Databricks for data engineering been so far?

Meet your classmates

- What are you hoping to get out of this class?

Getting Started with the Course



Course Objectives

1. Perform common code development tasks in a data engineering workflow using the Databricks **Data Science & Engineering Workspace**.
2. Use **Spark** to extract data from a variety of sources, apply common cleaning transformations, and manipulate complex data to load into **Delta Lake**.
3. Define and schedule data pipelines that incrementally ingest and process data through multiple tables in the lakehouse using **Delta Live Tables**.
4. Orchestrate data pipelines with Databricks **Workflow Jobs** and schedule dashboard updates to keep analytics up-to-date.
5. Configure permissions in **Unity Catalog** to ensure that users have proper access to databases for analytics and dashboarding.

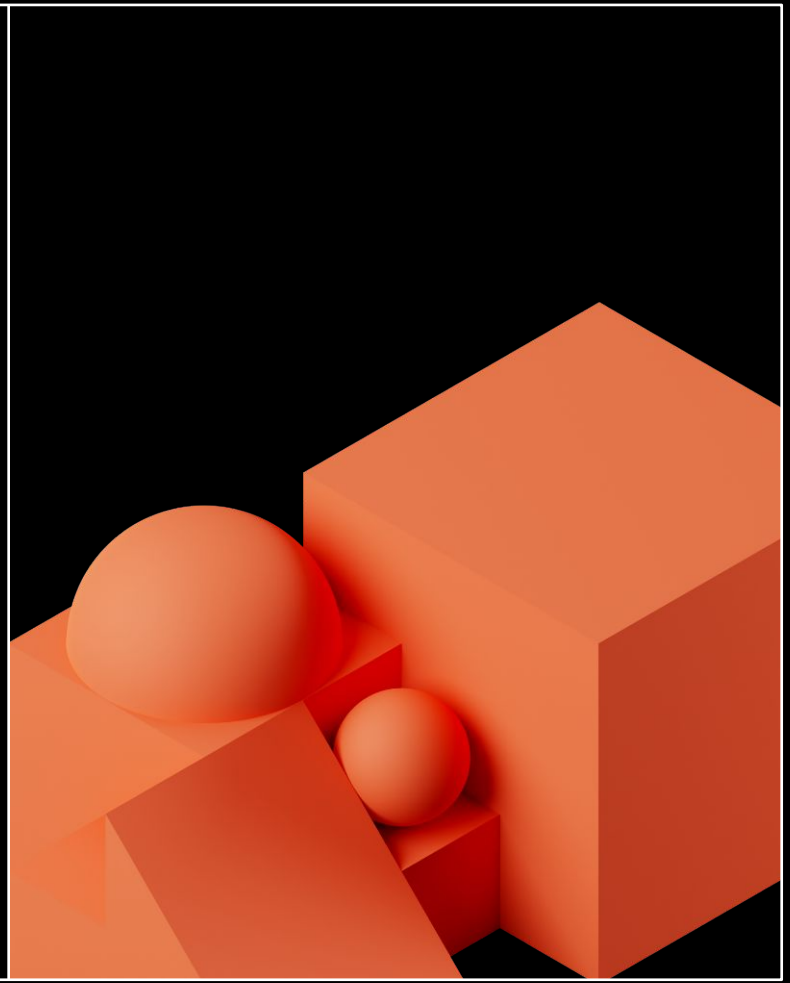
Agenda

Module Name	Duration
Get Started with Databricks Data Science and Engineering Workspace	1 hour, 20 min
Transform Data with Spark (SQL/PySpark)	2 hours, 50 min
Manage Data with Delta Lake	1 hour, 30 min
Build Data Pipelines with Delta Live Tables (SQL/PySpark)	3 hours
Deploy Workloads with Databricks Workflows	1 hour, 10 min
Manage Data Access for Analytics with Unity Catalog	2 hours

- We will take 10 minute breaks about every hour

Get Started with Databricks Data Science & Engineering Workspace

Module 01



Module Objectives

Get Started with Databricks Data Science and Engineering Workspace

1. Describe the core components of the Databricks Lakehouse platform.
2. Navigate the Databricks Data Science & Engineering Workspace UI.
3. Create and manage clusters using the Databricks Clusters UI.
4. Develop and run code in multi-cell Databricks notebooks using basic operations.
5. Integrate git support using Databricks Repos.

Module Overview

Get Started with Databricks Data Science and Engineering Workspace

Databricks Workspace and Services

Navigate the Workspace UI

Compute Resources

DE 1.1 - Create and Manage Interactive Clusters

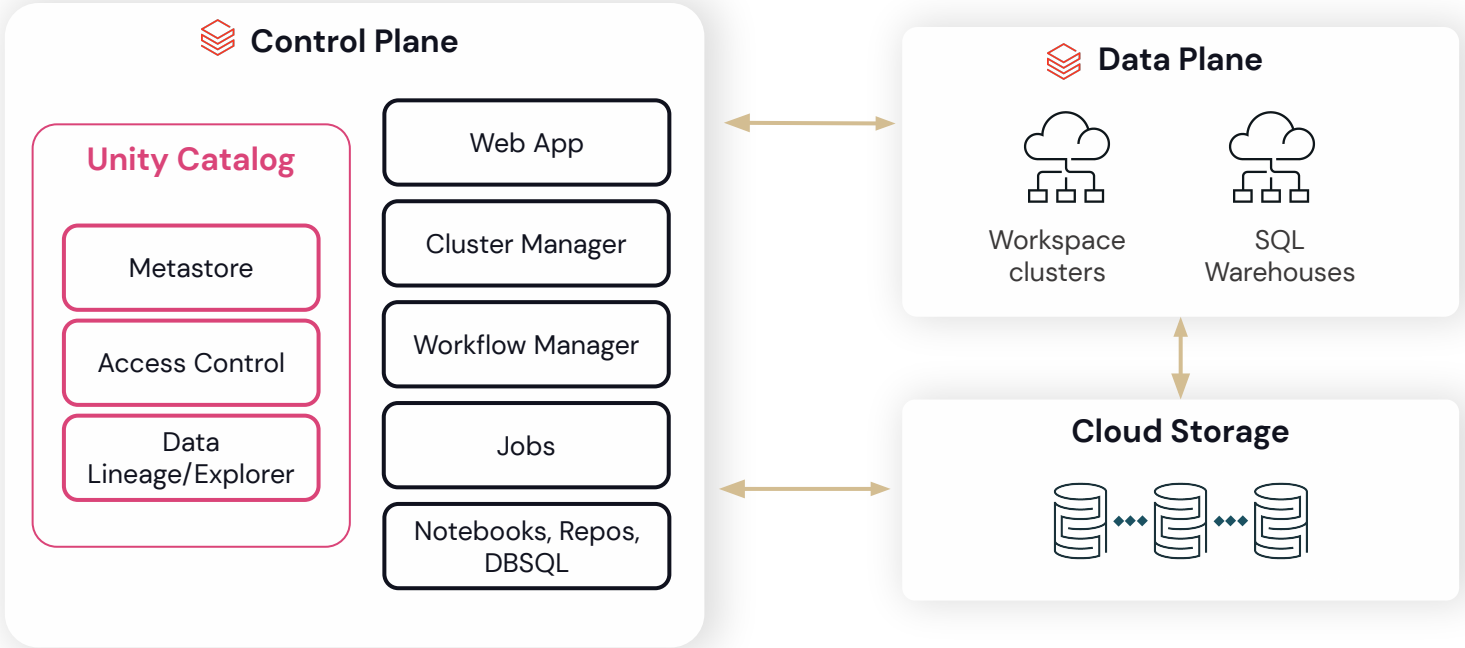
Develop Code with Notebooks & Databricks Repos

DE 1.2 - Databricks Notebook Operations

DE 1.3L - Get Started with the Databricks Platform Lab

Databricks Workspace and Services

Databricks Workspace and Services



Demo:
Navigate the Workspace UI

Compute Resources

Clusters

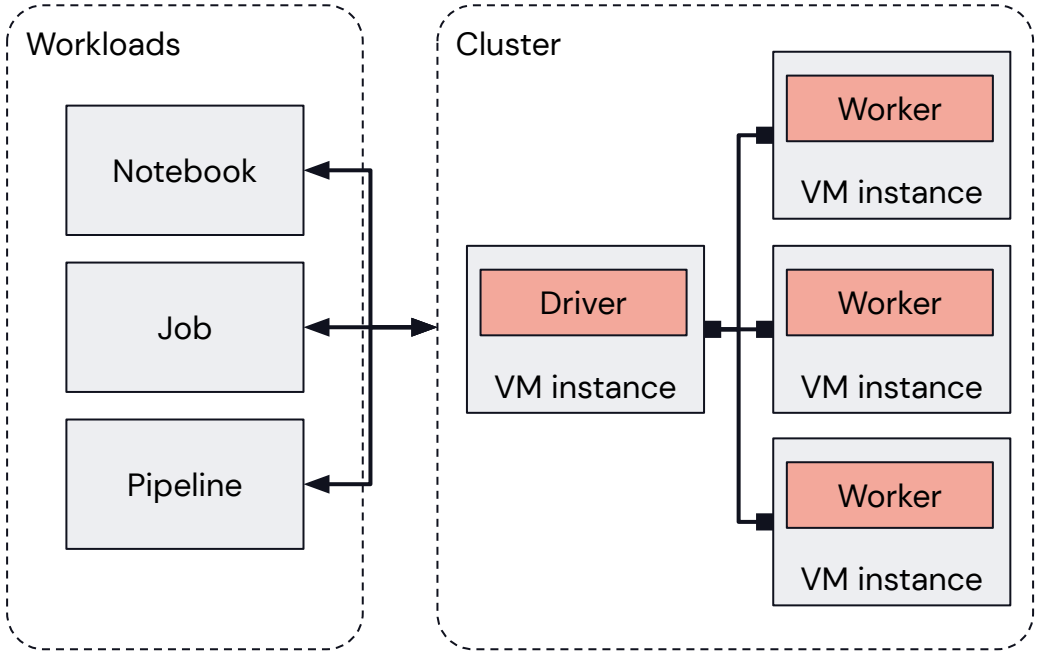
Overview

Collection of VM instances

Distributes workloads across workers

Two main types:

1. **All-purpose** clusters for interactive development
2. **Job** clusters for automating workloads



Cluster Types

All-purpose Clusters

Analyze data collaboratively using **interactive** notebooks

Create clusters from the Workspace or API

Configuration information retained for up to 70 clusters for up to 30 days

Job Clusters

Run **automated** jobs

The Databricks job scheduler creates job clusters when running jobs

Configuration information retained for up to 30 most recently terminated clusters

Cluster Configuration

Cluster Mode

Standard (Multi Node)

Default mode for workloads developed in any supported language (requires at least two VM instances)

Single node

Low-cost single-instance cluster catering to single-node machine learning workloads and lightweight exploratory analysis

Databricks Runtime Version

Standard

Apache Spark and many other components and updates to provide an optimized big data analytics experiences

Machine learning

Adds popular machine learning libraries like TensorFlow, Keras, PyTorch, and XGBoost.

Photon

An optional add-on to optimize SQL workloads

Access Mode

Access mode dropdown	Visible to user	Unity Catalog support	Supported languages
Single user	Always	Yes	Python, SQL, Scala, R
Shared	Always (Premium plan required)	Yes	Python (DBR 11.1+), SQL
No isolation shared	Can be hidden by enforcing user isolation in the admin console or configuring account-level settings	No	Python, SQL, Scala, R
Custom	Only shown for existing clusters *without* access modes (i.e. legacy cluster modes, Standard or High Concurrency); not an option for creating new clusters.	No	Python, SQL, Scala, R

Cluster Policies

Cluster policies can help to achieve the following:

- Standardize cluster configurations
- Provide predefined configurations targeting specific use cases
- Simplify the user experience
- Prevent excessive use and control cost
- Enforce correct tagging

Cluster Access Control

	No Permissions	Can Attach To	Can Restart	Can Manage
Attach notebook		✓	✓	✓
View Spark UI, cluster metrics, driver logs		✓	✓	✓
Start, restart, terminate			✓	✓
Edit				✓
Attach library				✓
Resize				✓
Change permissions				✓

Demo:
DE 1.1: Create and Manage Interactive Clusters

Develop Code with Notebooks & Databricks Repos

Databricks Notebooks

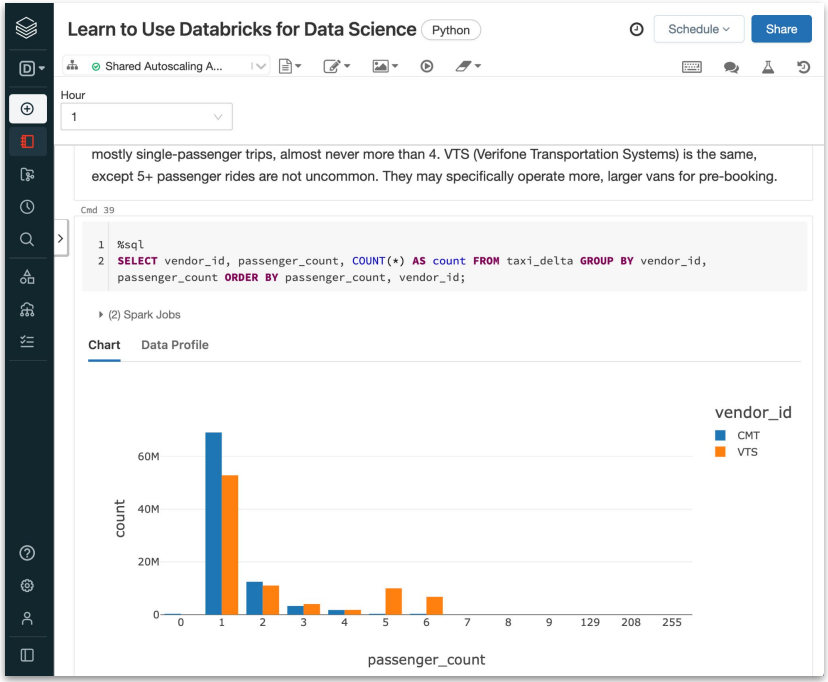
Collaborative, reproducible, and enterprise ready

Multi-language
Use Python, SQL, Scala, and R, all in one Notebook

Collaborative
Real-time co-presence, co-editing, and commenting

Ideal for exploration
Explore, visualize, and summarize data with built-in charts and data profiles

Adaptable
Install standard libraries and use local modules



Reproducible
Automatically track version history, and use git version control with Repos

Get to production faster
Quickly schedule notebooks as jobs or create dashboards from their results, all in the Notebook

Enterprise-ready
Enterprise-grade access controls, identity management, and auditability

Notebook magic commands

Use to override default languages, run utilities/auxiliary commands, etc.

%python, %r, %scala, %sql	Switch languages in a command cell
%sh	Run shell code (only runs on driver node, not worker nodes)
%fs	Shortcut for `dbutils` filesystem commands
%md	Markdown for styling the display
%run	Execute a remote notebook from a notebook
%pip	Install new Python libraries

dbutils (Databricks Utilities)

Perform various tasks with Databricks using notebooks

Utility	Description	Example
fs	Manipulates the Databricks filesystem (DBFS) from the console	dbutils.fs.ls()
secrets	Provides utilities for leveraging secrets within notebooks	dbutils.secrets.get()
notebook	Utilities for the control flow of a notebook	dbutils.notebook.run()
widgets	Methods to create and get bound value of input widgets inside notebooks	dbutils.widget.text()
jobs	Utilities for leveraging jobs features	dbutils.jobs.taskValues.set()

Available within Python, R, or Scala notebooks

Git Versioning with Databricks Repos

Databricks Repos

Git Versioning

Native integration with Github, Gitlab, Bitbucket and Azure Devops

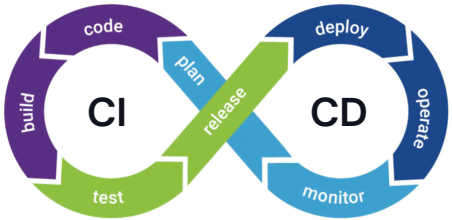
UI-based workflows

Azure DevOps GitHub GitLab Bitbucket

CI/CD Integration

API surface to integrate with automation

Simplifies the dev/staging/prod multi-workspace story



Enterprise ready

Allow lists to avoid exfiltration

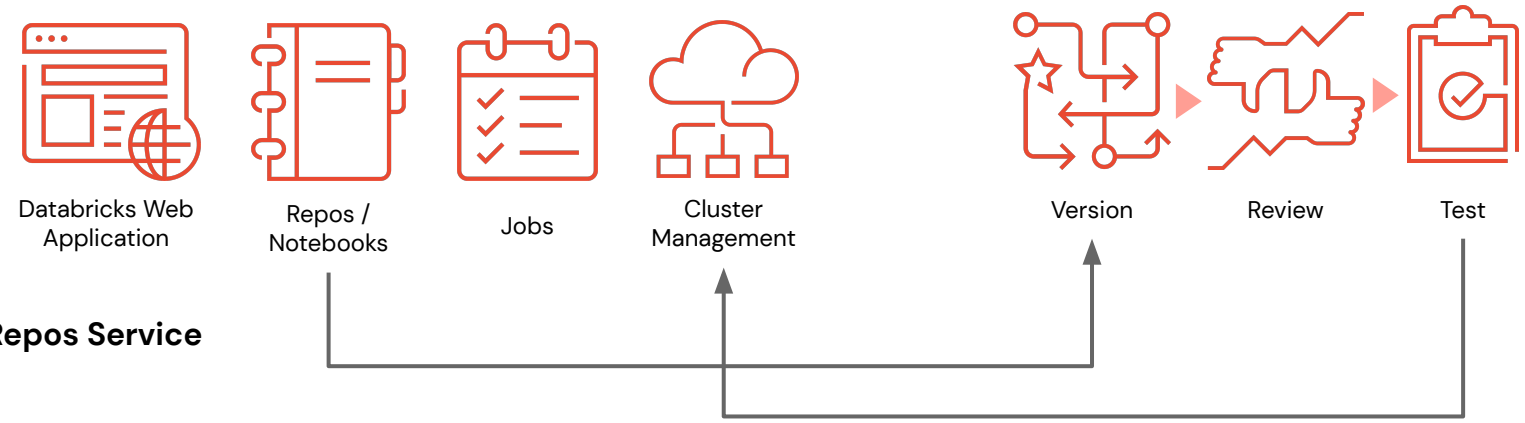
Secret detection to avoid leaking keys

Databricks Repos

CI/CD Integration

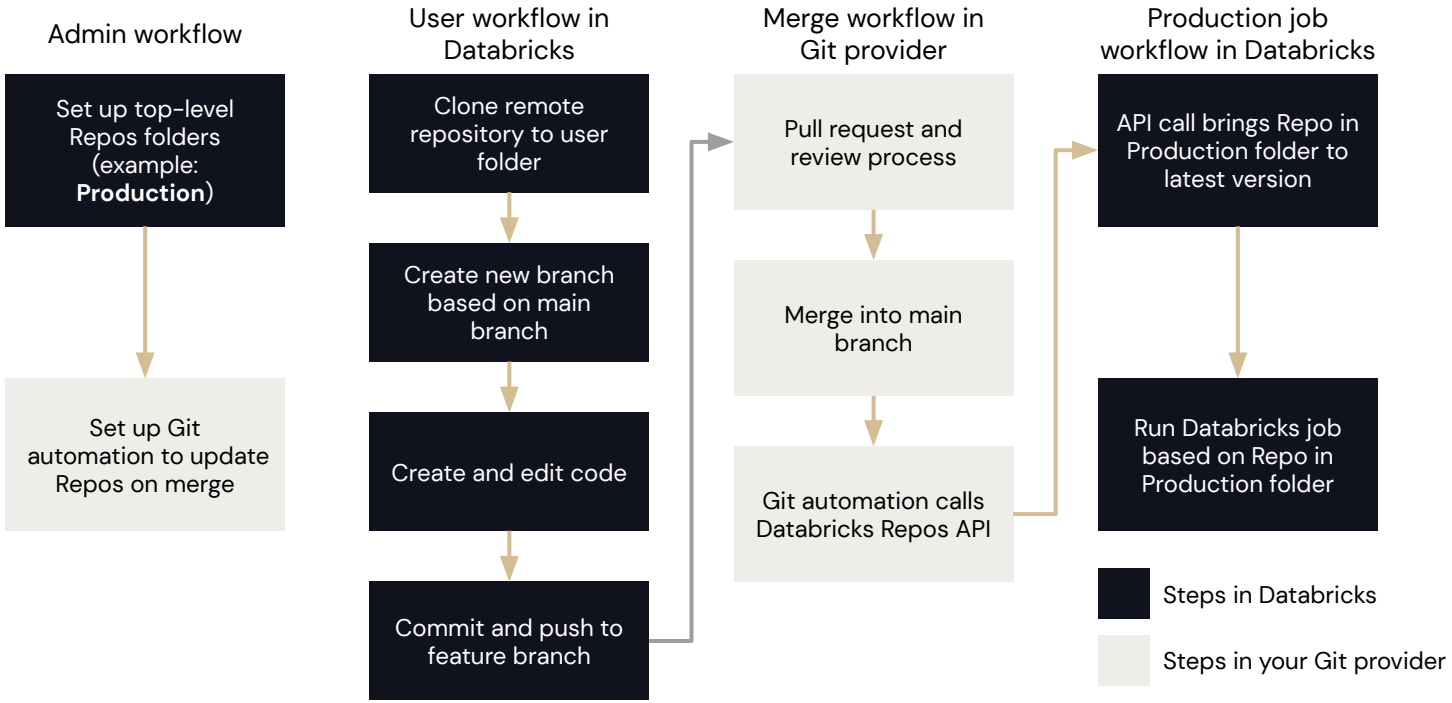
Control Plane in Databricks
Manage customer accounts, datasets, and clusters

Git and CI/CD Systems



CI/CD workflows with Git and Repos

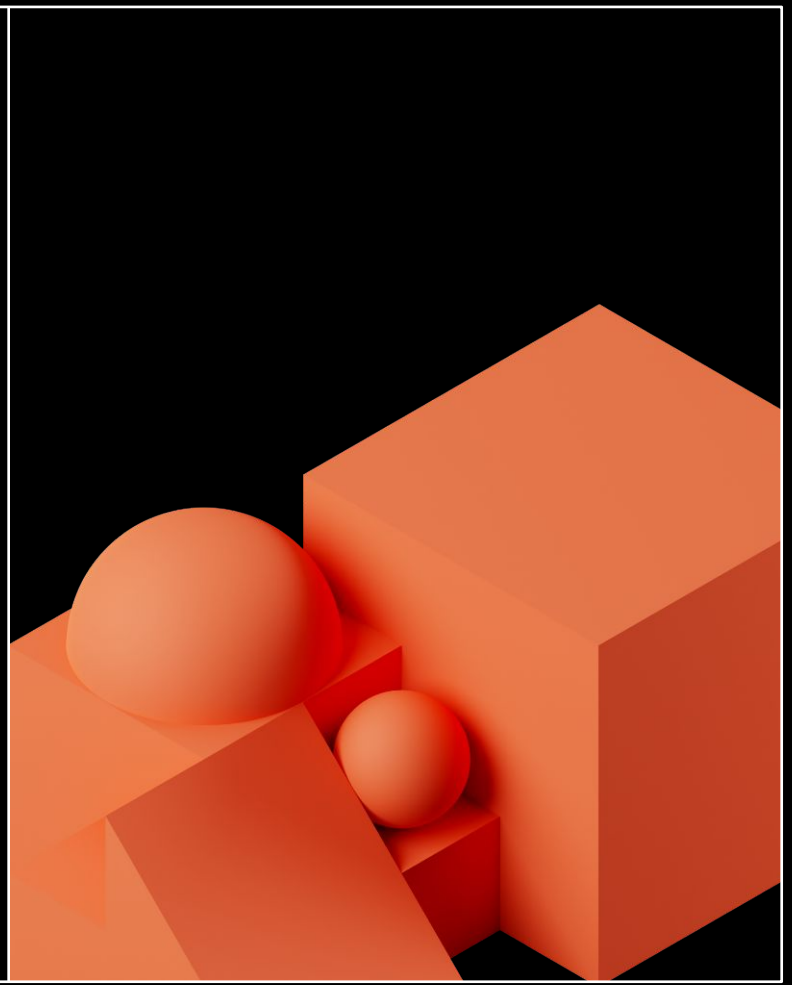
Documentation



Demo:
DE 1.2: Databricks Notebook Operations

Lab:
DE 1.3L: Get Started with the Databricks Platform

Transform Data with Spark



Module Objectives

Transform Data with Spark

1. Extract data from a variety of file formats and data sources using Spark
2. Apply a number of common transformations to clean data using Spark
3. Reshape and manipulate complex data using advanced built-in functions in Spark
4. Leverage UDFs for reusable code and apply best practices for performance in Spark

Module Agenda

Transform Data with Spark

Data Objects in the Lakehouse

DE 2.1 - Querying Files Directly

DE 2.2 - Options for External Sources

DE 2.3L - Extract Data Lab

DE 2.4 - Cleaning Data

DE 2.5 - Complex Transformations

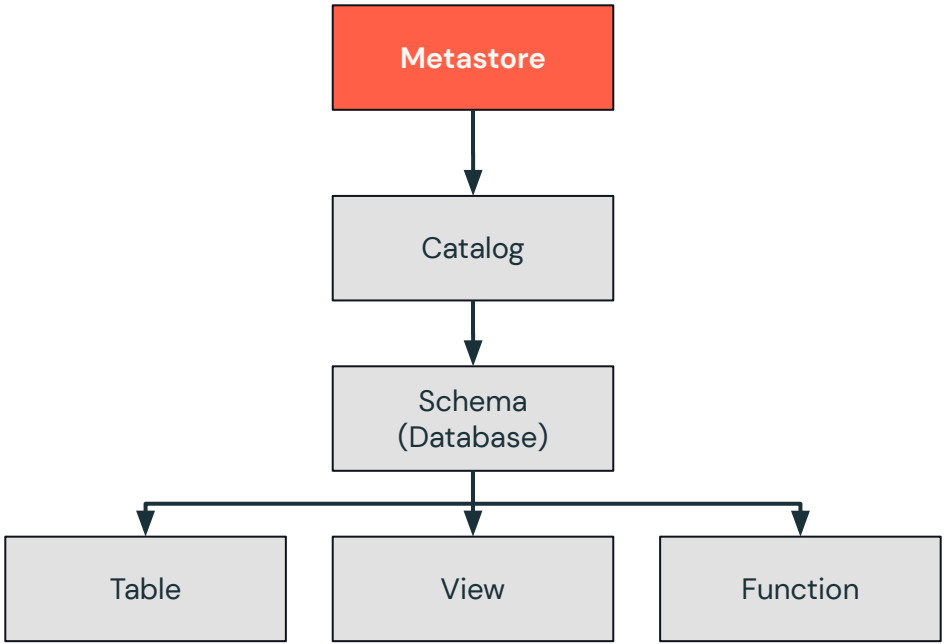
DE 2.6L - Reshape Data Lab

DE 2.7A – SQL UDFs and Control Flow

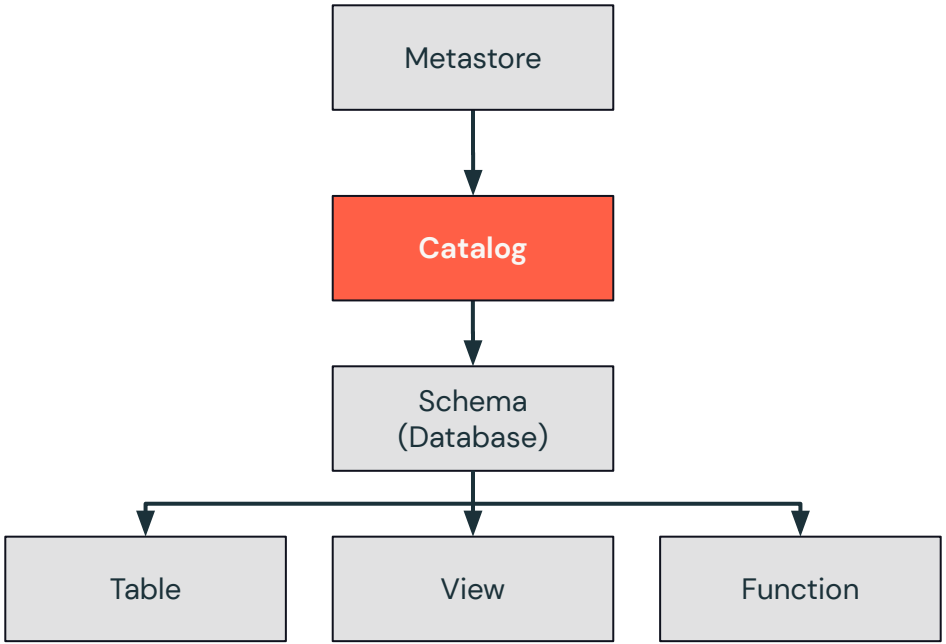
DE 2.7B - Python UDFs

Data Objects in the Lakehouse

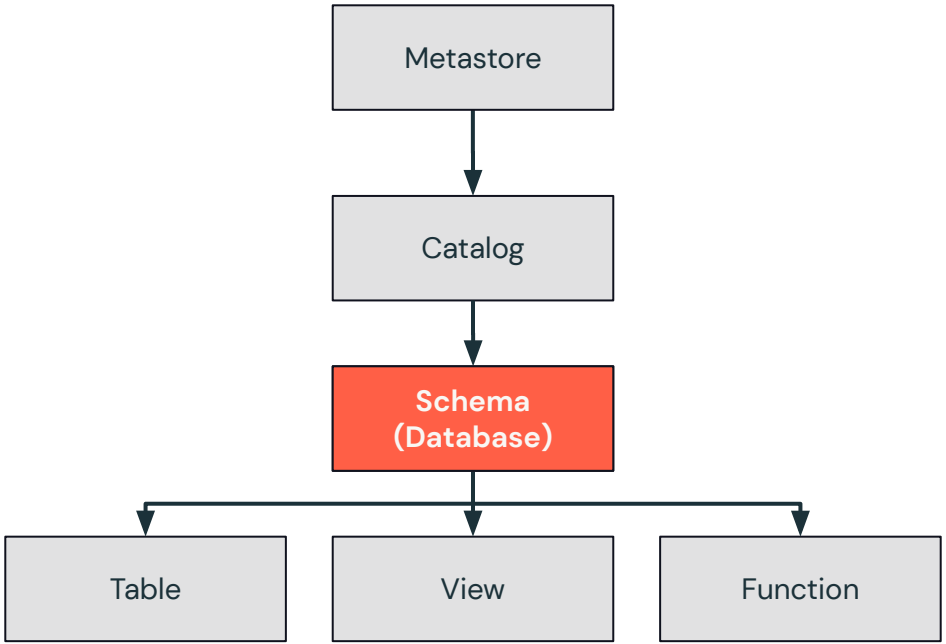
Data objects in the Lakehouse



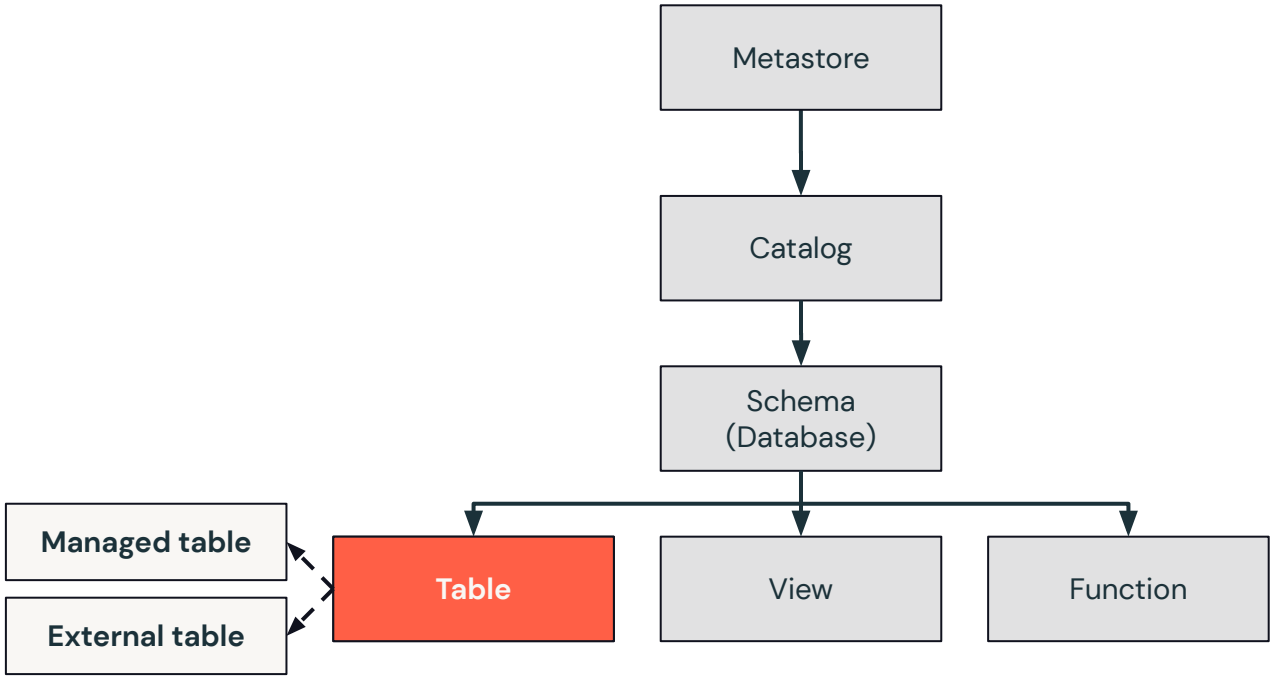
Data objects in the Lakehouse



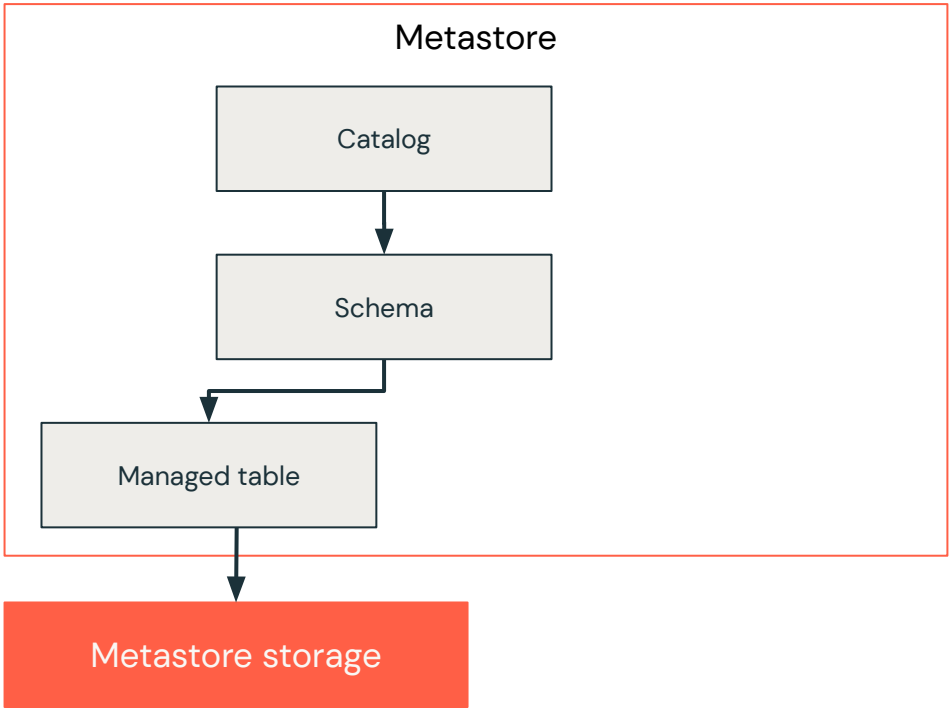
Data objects in the Lakehouse



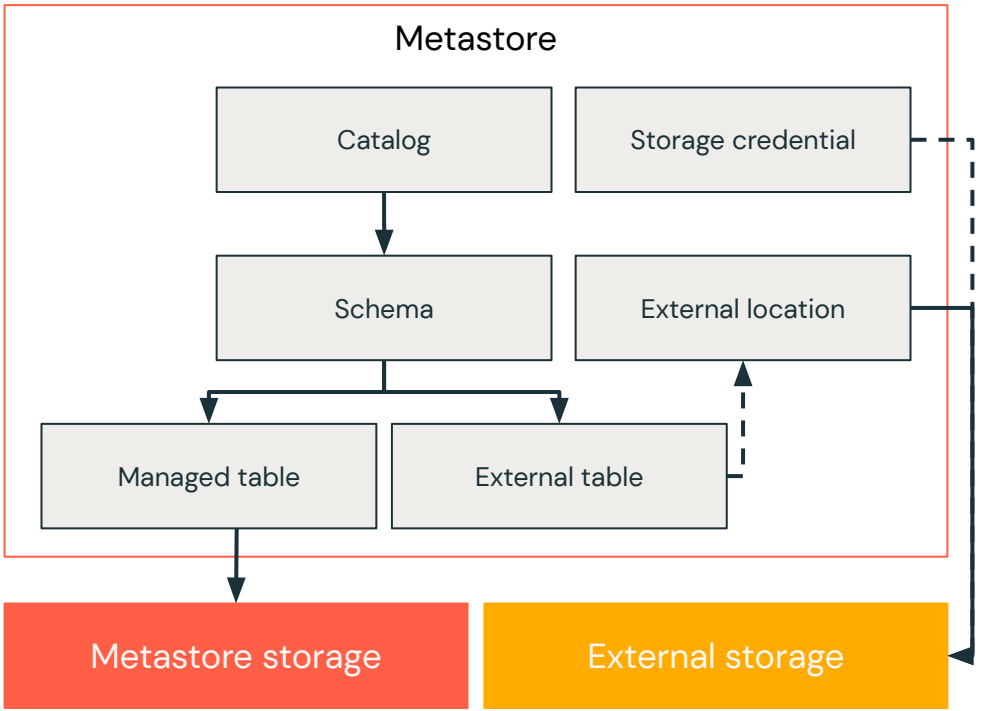
Data objects in the Lakehouse



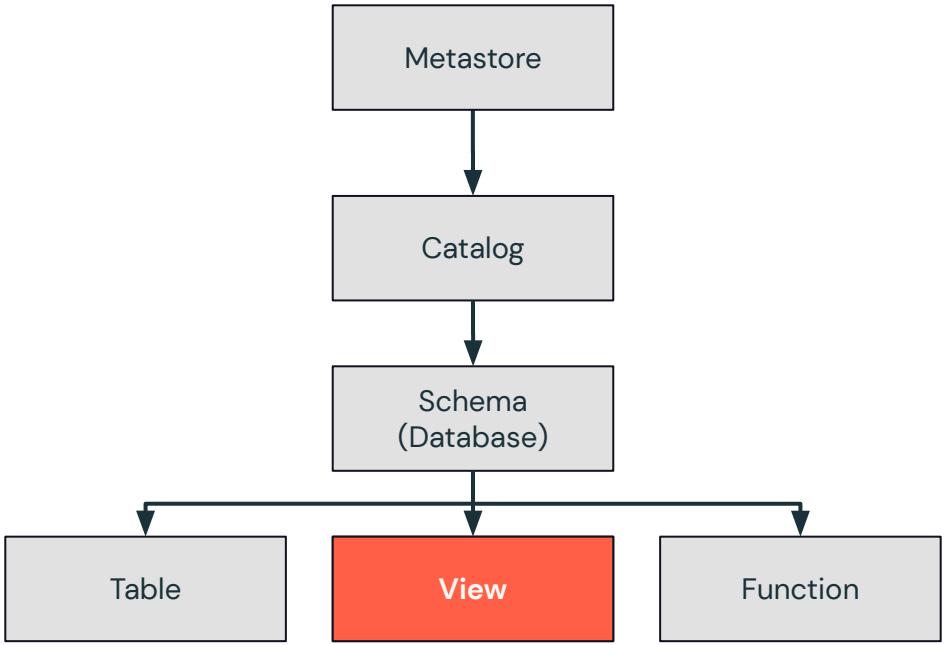
Managed Tables



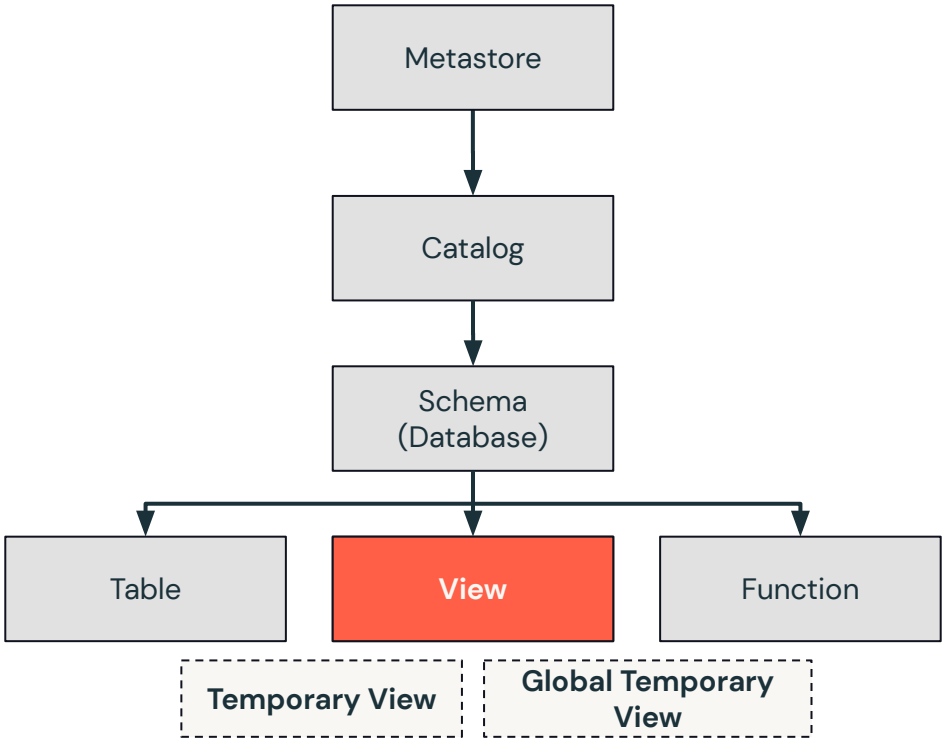
External Tables



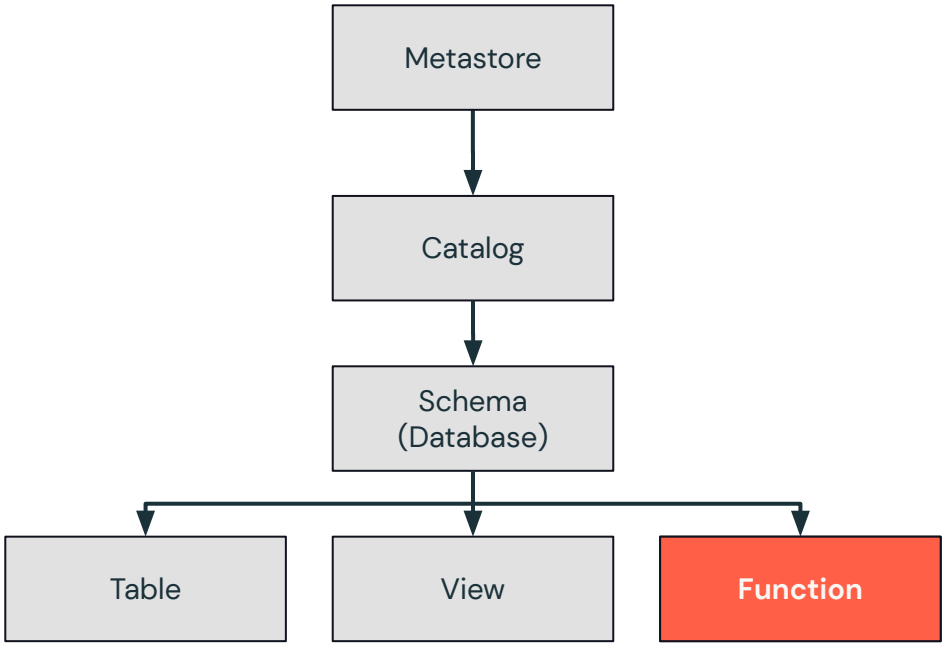
Data objects in the Lakehouse



Data objects in the Lakehouse



Data objects in the Lakehouse



Extracting Data

Query files directly

`SELECT * FROM file_format.`path/to/file``

Files can be queried directly using SQL

- `SELECT * FROM json.`path/to/files/``
- `SELECT * FROM text.`path/to/files/``

Process based on specified file format

- `json` pulls schema from underlying data
- `binaryFile` and `text` file formats have fixed data schemas
  - `text` → string `value` column (row for each line)
  - `binaryFile` → `path`, `modificationTime`, `length`, `content` columns (row for each file)

Configure external tables with read options

`CREATE TABLE USING data_source OPTIONS (...)`

Many data sources require schema declaration and other options to correctly read data

- CSV options for delimiter, header, etc
- JDBC options for url, user, password, etc
  - Note: using the JDBC driver pulls RDBMS tables dynamically for Spark processing

Demo:
DE 2.1: Querying Files Directly

Demo:
DE 2.2: Providing Options for External Sources

Lab:
DE 2.3L: Extract Data Lab

Lab:
DE 2.4: Cleaning Data

Complex Transformations

Interact with Nested Data

Use built-in syntax to traverse nested data with Spark SQL

Use “ : ” (colon) syntax in queries to access subfields in JSON strings

```
SELECT value:device, value:geo ...
```

Use “ . ” (dot) syntax in queries to access subfields in STRUCT types

```
SELECT value.device, value.geo ...
```


Complex Types

Nested data types storing multiple values

- **Array**: arbitrary number of elements of same data type
- **Map**: set of key-value pairs
- **Struct**: ordered (fixed) collection of column(s) and any data type

Example table with complex types

```
CREATE TABLE employees (name STRING, salary FLOAT,
   subordinates ARRAY<STRING>,
   deductions   MAP<STRING, FLOAT>,
   address      STRUCT<street:STRING,city:STRING,state:STRING, zip:INT>)
```


Demo:
DE 2.5: Complex Transformations

explode lab

```
SELECT
   user_id, event_timestamp, event_name,
   explode(items) AS item
FROM events
```

`explode` outputs the elements of an array field into a separate row for each element

user_id	event_timestamp	event_name	items
UA000000106494077	1593612846854930	add_item	[{"coupon": null, "item_id": "M_PREM_Q", "item_name": "Premium Queen Mattress", "item_revenue_in_usd": 1795, "price_in_usd": 1795, "qua... {"coupon": null, "ite... "M_STAN_Q", "item_name": "Standard Queen Mattress", "item_revenue_in_usd": 1... price_in_usd": 1045, "qu... {"coupon": null, "item_id": "M_STAN_T", "item_name": "Standard Twin Mattress", "item_revenue_in_usd": 595, "price_in_usd": 595, "quantity": 1}]

Each item in the `items` array above is exploded into its own row, resulting in the 3 rows below

user_id	event_timestamp	event_name	item
UA000000106494077	1593612846854930	add_ite...	{"coupon": null, "item_id": "M_PREM_Q", "item_name": "Premium Queen Mattress", "item_revenue_in_usd": 1795, "price_in_usd": 1795, "quantity": 1}
UA000000106494077	1593612846854930	add_ite...	{"coupon": null, "item_id": "M_STAN_Q", "item_name": "Standard Queen Mattress", "item_revenue_in_usd": 1045, "price_in_usd": 1045, "quantity": 1}
UA000000106494077	1593612846854930	add_ite...	{"coupon": null, "item_id": "M_STAN_T", "item_name": "Standard Twin Mattress", "item_revenue_in_usd": 595, "price_in_usd": 595, "quantity": 1}

flatten lab

`collect_set` returns an array of unique values from a field for each group of rows

`flatten` returns an array that flattens multiple arrays into one

```
SELECT user_id,
  collect_set(event_name) AS event_history,
  array_distinct(flatten(collect_set(items.item_id))) AS cart_history
FROM events
GROUP BY user_id
```

user_id	event_name	items
UA000000106494077	add_item	[{"coupon": null, "item_id": "M_PREM_Q", "item_name": "Premium Queen Mattress", "item_revenue_in_usd": 1795, "price_in_usd": 1795, "quantity": 1}, {"coupon": null, "item_id": "M_STAN_Q", "item_name": "Standard Queen Mattress", "item_revenue_in_usd": 1045, "price_in_usd": 1045, "quantity": 1}]
UA000000106494077	delivery	[]
UA000000106494077	email_coupon	[{"coupon": null, "item_id": "M_PREM_Q", "item_name": "Premium Queen Mattress", "item_revenue_in_usd": 1795, "price_in_usd": 1795, "quantity": 1}, {"coupon": null, "item_id": "M_STAN_Q", "item_name": "Standard Queen Mattress", "item_revenue_in_usd": 1045, "price_in_usd": 1045, "quantity": 1}, {"coupon": null, "item_id": "M_STAN_T", "item_name": "Standard Twin Mattress", "item_revenue_in_usd": 595, "price_in_usd": 595, "quantity": 1}]
UA000000106494077	main	[]
UA000000106494077	original	[]
UA000000106494077	premium	[{"coupon": null, "item_id": "M_PREM_Q", "item_name": "Premium Queen Mattress", "item_revenue_in_usd": 1795, "price_in_usd": 1795, "quantity": 1}, {"coupon": null, "item_id": "M_STAN_Q", "item_name": "Standard Queen Mattress", "item_revenue_in_usd": 1045, "price_in_usd": 1045, "quantity": 1}, {"coupon": null, "item_id": "M_STAN_T", "item_name": "Standard Twin Mattress", "item_revenue_in_usd": 595, "price_in_usd": 595, "quantity": 1}]
UA000000106494077	reviews	[]

	user_id	event_history	cart_history
1	UA000000106494077	["add_item", "email_coupon", "main", "reviews", "original", "delivery", "premium"]	["M_PREM_Q", "M_STAN_Q", "M_STAN_T"]

Collection example

`collect_set` returns an array with duplicate elements eliminated

`collect_list` returns an array with duplicate elements intact

`df`

age
2
5
5

`df.agg(collect_set('age'))`

collect_set(age)
▸ [5, 2]

`df.agg(collect_list('age'))`

collect_list(age)
▸ [2, 5, 5]

Parse JSON strings into structs

Create the schema to parse the JSON strings by providing an example JSON string from a row that has no nulls

`from_json` uses JSON schema returned by `schema_of_json` to convert a column of JSON strings into structs

This highlighted JSON string is taken from the `value` field of a single row of data

```
CREATE OR REPLACE TABLE parsed_events AS
  SELECT from_json(value, schema_of_json('{"device":"Linux","ecommerce":
{"purchase_revenue_in_usd":1075.5,"total_item_quantity":1,"unique_items":1},"event_name":"finalize","event_previous_timestamp":1
593879231210816,"event_timestamp":1593879335779563,"geo":{"city":"Houston","state":"TX"},"items":
[{"coupon":"NEWBED10","item_id":"M_STAN_K","item_name":"Standard King
Mattress","item_revenue_in_usd":1075.5,"price_in_usd":1195.0,"quantity":1}],"traffic_source":"email","user_first_touch_timestamp
":1593454417513109,"user_id":"UA000000106116176"}')) AS new_struct
  FROM events_strings;
```

col_name	data_type
new_struct	struct<device:string,ecommerce:struct<purchase_revenue_in_usd:double,total_item_quantity:bigint,unique_items:bigint>,event_name:string,event_previous_timestamp:bigint,event_timestamp:bigint,geo:struct<city:string,state:string>,items:array<struct<coupon:string,item_id:string,item_name:string,item_revenue_in_usd:double,price_in_usd:double,quantity:bigint>>,traffic_source:string,user_first_touch_timestamp:bigint,user_id:string>

Returns STRUCT column containing ARRAY of nested STRUCT

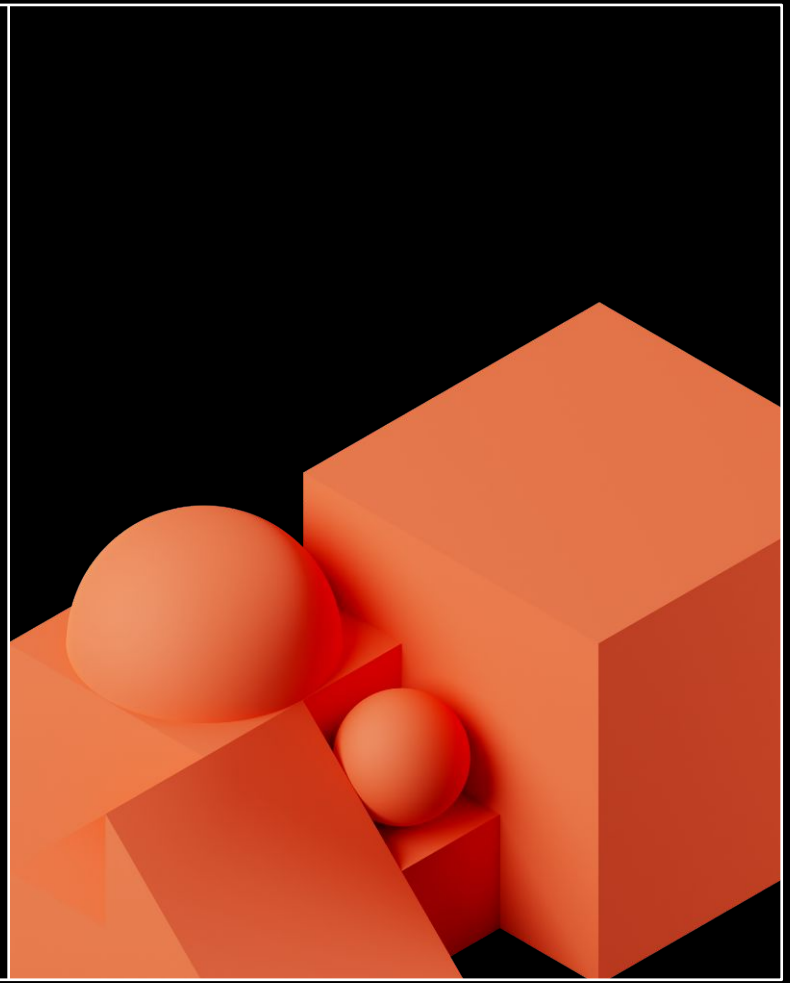
Lab:
DE 2.5L: Reshape Data Lab (Optional)

Demo:
DE 2.7A: SQL UDFs and Control Flow (Optional)

Demo:
DE 2.7B: Python UDFs (Optional)

Manage Data with Delta Lake

Module 03



Module Agenda

Manage Data with Delta Lake

What is Delta Lake

DE 3.1 - Schemas and Tables

DE 3.2 - Version and Optimize Delta Tables

DE 3.3L - Manipulate Delta Tables Lab

DE 3.4 - Set Up Delta Tables

DE 3.5 - Load Data into Delta Lake

DE 3.6L - Load Data Lab

What is Delta Lake?

Delta Lake is an open-source project that enables building a data lakehouse on top of existing cloud storage

Delta Lake Is Not...

- Proprietary technology
- Storage format
- Storage medium
- Database service or data warehouse

Delta Lake Is...

- Open source
- Builds upon standard data formats
- Optimized for cloud object storage
- Built for scalable metadata handling

Delta Lake brings ACID to object storage

Atomicity means all transactions either succeed or fail completely

Consistency guarantees relate to how a given state of the data is observed by simultaneous operations

Isolation refers to how simultaneous operations conflict with one another. The isolation guarantees that Delta Lake provides do differ from other systems

Durability means that committed changes are permanent

Problems solved by ACID

- Hard to append data
- Modification of existing data difficult
- Jobs failing mid way
- Real-time operations hard
- Costly to keep historical data versions

Delta Lake is the default format for tables created in Databricks

```
CREATE TABLE foo
USING DELTA
```

```
df.write
  .format("delta")
```

Demo:
DE 3.1: Schemas and Tables

Demo:
DE 3.2: Version and Optimize Delta Tables

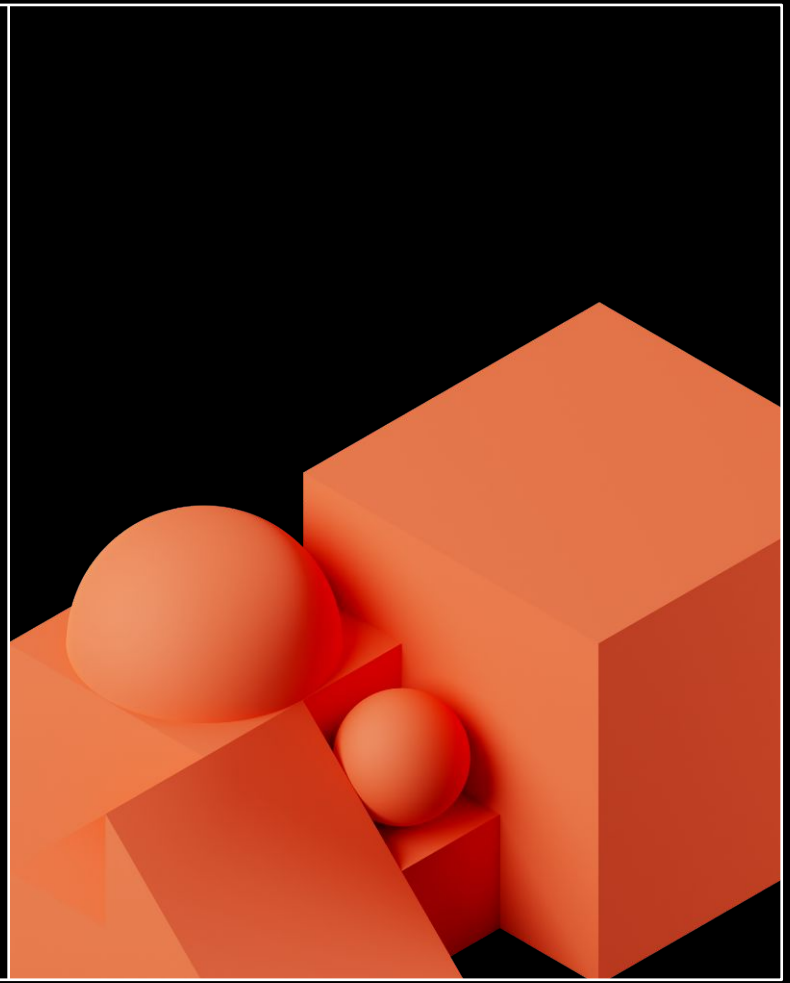
Lab:
DE 3.3L – Manipulate Delta Tables Lab

Demo:
DE 3.4: Set up Delta Tables

Demo:
DE 3.5: Load Data into Delta Tables

Lab:
DE 3.6L: Load Data

Build Data Pipelines with Delta Live Tables



Agenda

Build Data Pipelines with Delta Live Tables

The Medallion Architecture

Introduction to Delta Live Tables

DE 4.1 - DLT UI Walkthrough

DE 4.1A - SQL Pipelines

DE 4.1B - Python Pipelines

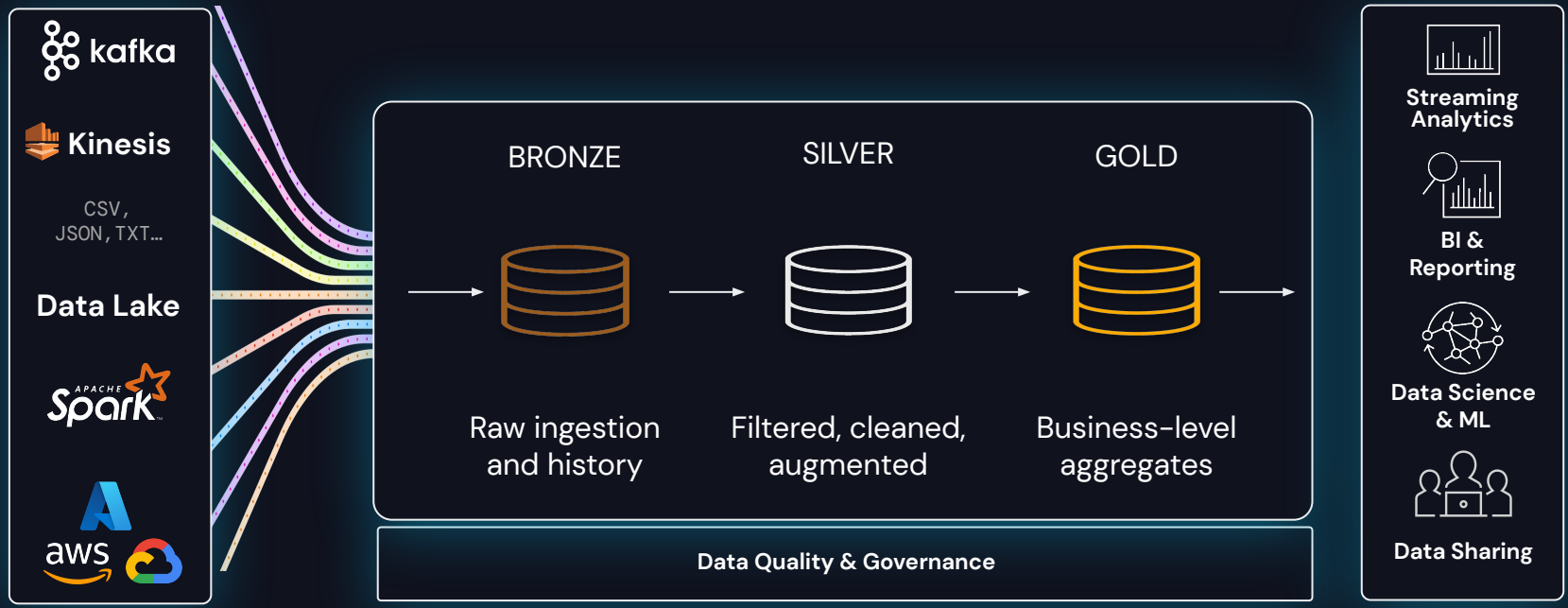
DE 4.2 - Python vs SQL

DE 4.3 - Pipeline Results

DE 4.4 - Pipeline Event Logs

The Medallion Architecture

Medallion Architecture in the Lakehouse



Multi-Hop in the Lakehouse

Bronze Layer

Typically just a raw copy of ingested data

Replaces traditional data lake

Provides efficient storage and querying of full, unprocessed history of data

Multi-Hop in the Lakehouse

Silver Layer

Reduces data storage complexity, latency, and redundancy

Optimizes ETL throughput and analytic query performance

Preserves grain of original data (without aggregations)

Eliminates duplicate records

Production schema enforced

Data quality checks, corrupt data quarantined

Multi-Hop in the Lakehouse

Gold Layer

Powers ML applications, reporting, dashboards, ad hoc analytics

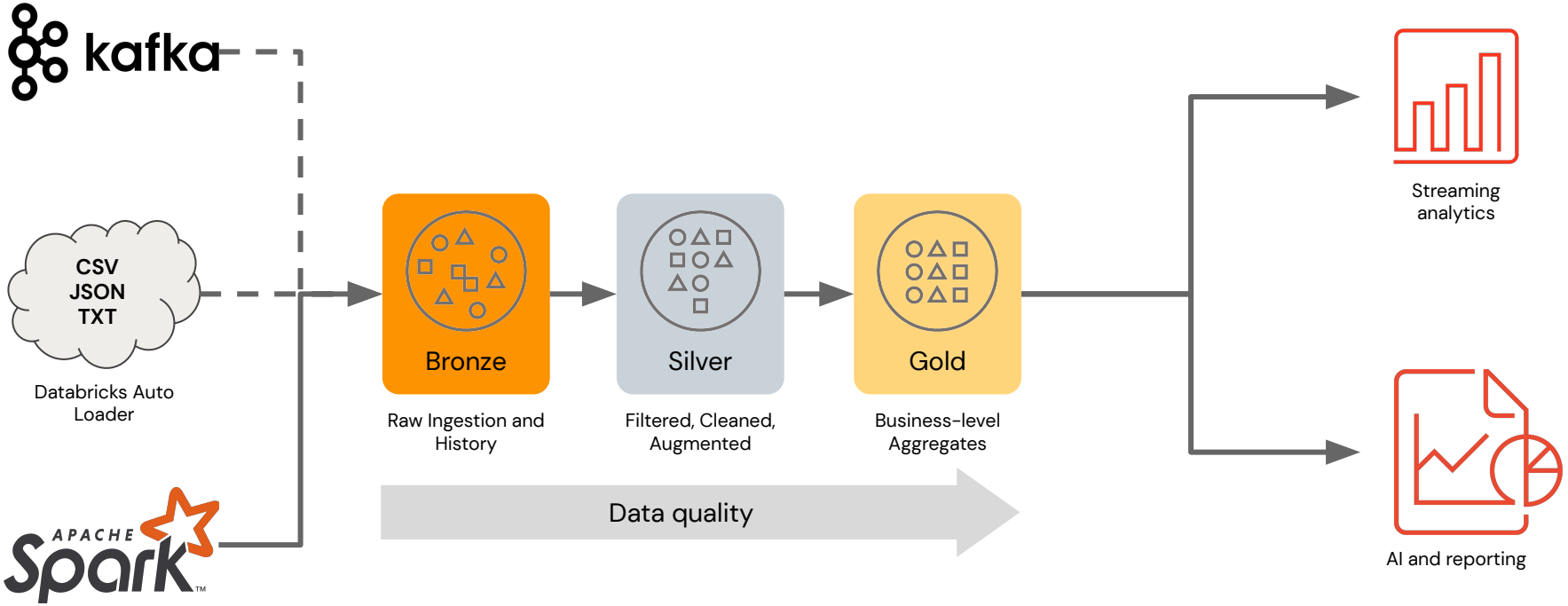
Refined views of data, typically with aggregations

Reduces strain on production systems

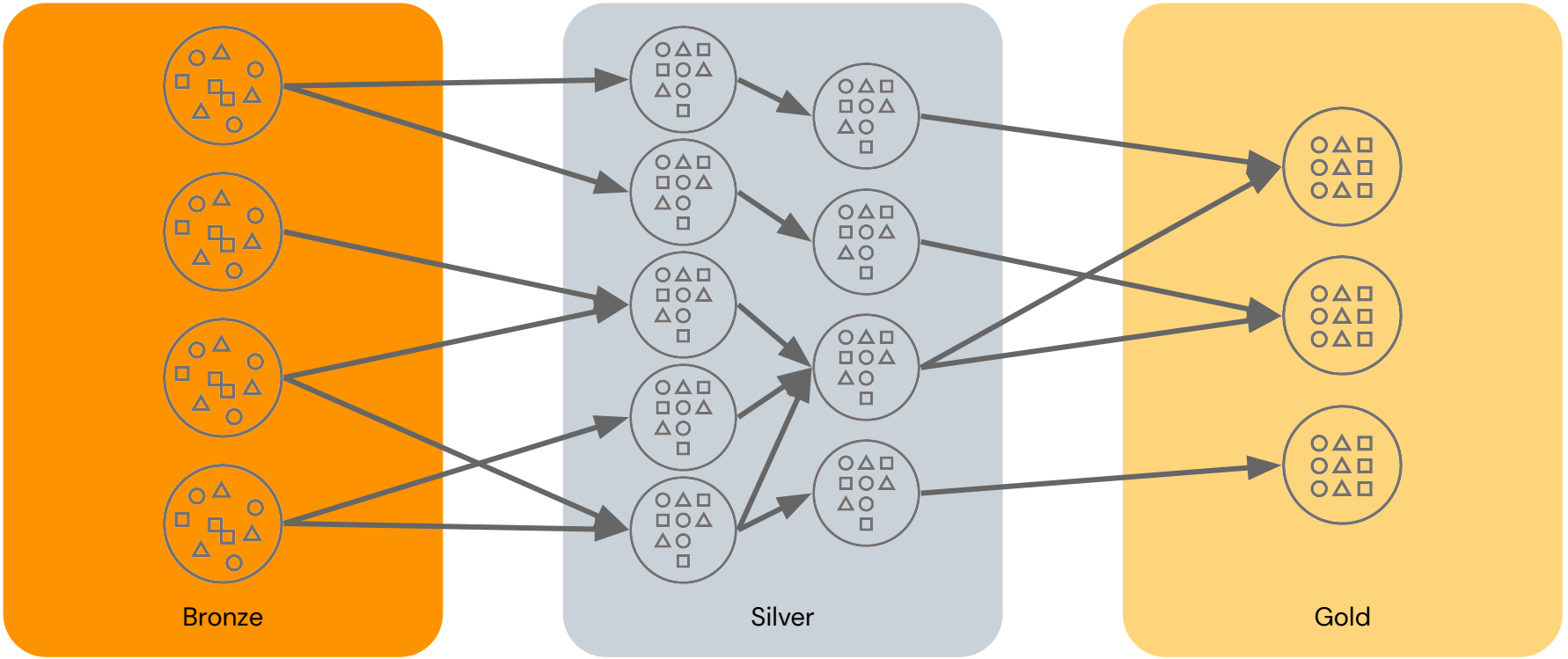
Optimizes query performance for business-critical data

Introduction to Delta Live Tables

Multi-Hop in the Lakehouse



The Reality is Not so Simple



Large scale ETL is complex and brittle

Complex pipeline development

Hard to build and maintain table **dependencies**

Difficult to switch between **batch** and **stream** processing

Data quality and governance

Difficult to monitor and enforce **data quality**

Impossible to trace data **lineage**

Difficult pipeline operations

Poor **observability** at granular, data level

Error handling and **recovery** is laborious

Introducing Delta Live Tables

Make reliable ETL easy on Delta Lake

Operate with agility

Declarative tools to build batch and streaming data pipelines

Trust your data

DLT has built-in declarative quality controls

Declare quality expectations and actions to take

Scale with reliability

Easily scale infrastructure alongside your data

What is a LIVE TABLE?

What is a Live Table?

Live Tables are materialized views for the lakehouse.

A live table is:

- Defined by a SQL query
- Created and kept up-to-date by a pipeline

```
CREATE OR REFRESH LIVE TABLE report
AS SELECT sum(profit)
FROM prod.sales
```

Live tables provides tools to:

- Manage dependencies
- Control quality
- Automate operations
- Simplify collaboration
- Save costs
- Reduce latency

What is a Streaming Live Table?

Based on Spark™ Structured Streaming

A streaming live table is "stateful":

- Ensures exactly-once processing of input rows
- Inputs are only read once

```
CREATE STREAMING LIVE TABLE report
AS SELECT sum(profit)
FROM cloud_files(prod.sales)
```

- Streaming Live tables compute results over append-only streams such as Kafka, Kinesis, or Auto Loader (files on cloud storage)
- Streaming live tables allow you to reduce costs and latency by avoiding reprocessing of old data.

When should I use streaming?

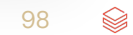
Using Spark Structured Streaming for ingestion

Easily ingest files from cloud storage as they are uploaded

```
CREATE STREAMING LIVE TABLE raw_data
AS SELECT *
FROM cloud_files("/data", "json")
```

This example creates a table with all the json data stored in "/data":

- `cloud_files` keeps track of which files have been read to avoid duplication and wasted work
- Supports both listing and notifications for arbitrary scale
- Configurable schema inference and schema evolution



Using the SQL STREAM( ) function

Stream data from any Delta table

```
CREATE STREAMING LIVE TABLE
mystream
  AS SELECT *
  FROM STREAM(my_table)
```

Pitfall: `my_table` must be an append-only source.

e.g. it may not:

- be the target of `APPLY CHANGES INTO`
- define an aggregate function
- be a table on which you've executed DML to delete/update a row (see GDPR section)

- `STREAM(my_table)` reads a stream of new records, instead of a snapshot
- `Streaming` tables must be an append-only table
- Any append-only delta table can be read as a stream (i.e. from the live schema, from the catalog, or just from a path).

How do I use DLT?

Creating Your First Live Table Pipeline

SQL to DLT in three easy steps…

Write create live table

- Table definitions are written (but not run) in notebooks
- Databricks Repos allow you to version control your table definitions.

```
CREATE LIVE TABLE daily_stats
AS SELECT sum(rev) - sum(costs) AS profits
FROM prod_data.transactions
GROUP BY day
```

Create a pipeline

- A Pipeline picks one or more notebooks of table definitions, as well as any configuration required.

Click start

- DLT will create or update all the tables in the pipelines.

BEST PRACTICE

Development vs Production

Fast iteration or enterprise grade reliability

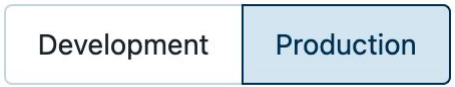
Development Mode

- Reuses a **long-running cluster** running for fast iteration.
- **No retries** on errors enabling faster debugging.

Production Mode

- Cuts costs by **turning off clusters** as soon as they are done (within 5 minutes)
- **Escalating retries**, including cluster restarts, ensure reliability in the face of transient issues.

In the Pipelines UI:

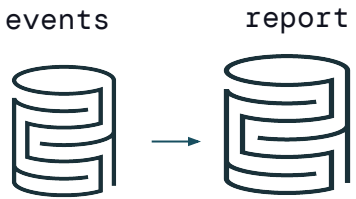


What if I have dependent tables?

Declare LIVE Dependencies

Using the LIVE virtual schema.

```
CREATE LIVE TABLE events
AS SELECT … FROM prod.raw_data

CREATE LIVE TABLE report
AS SELECT … FROM LIVE.events
```



- Dependencies owned by other producers are just read from the catalog or spark data source as normal.
- LIVE dependencies, from the **same pipeline**, are read from the LIVE schema.
- DLT detects LIVE dependencies and executes all operations in **correct order**.
- DLT handles parallelism and captures the lineage of the data.

How do I ensure Data Quality?

Ensure correctness with Expectations

Expectations are tests that ensure data quality in production

```
CONSTRAINT valid_timestamp
EXPECT (timestamp > '2012-01-01')
ON VIOLATION DROP
```

```
@dlt.expect_or_drop(
  "valid_timestamp",
  col("timestamp") > '2012-01-01')
```

Expectations are true/false expressions that are used to validate each row during processing.

DLT offers flexible policies on how to handle records that violate expectations:

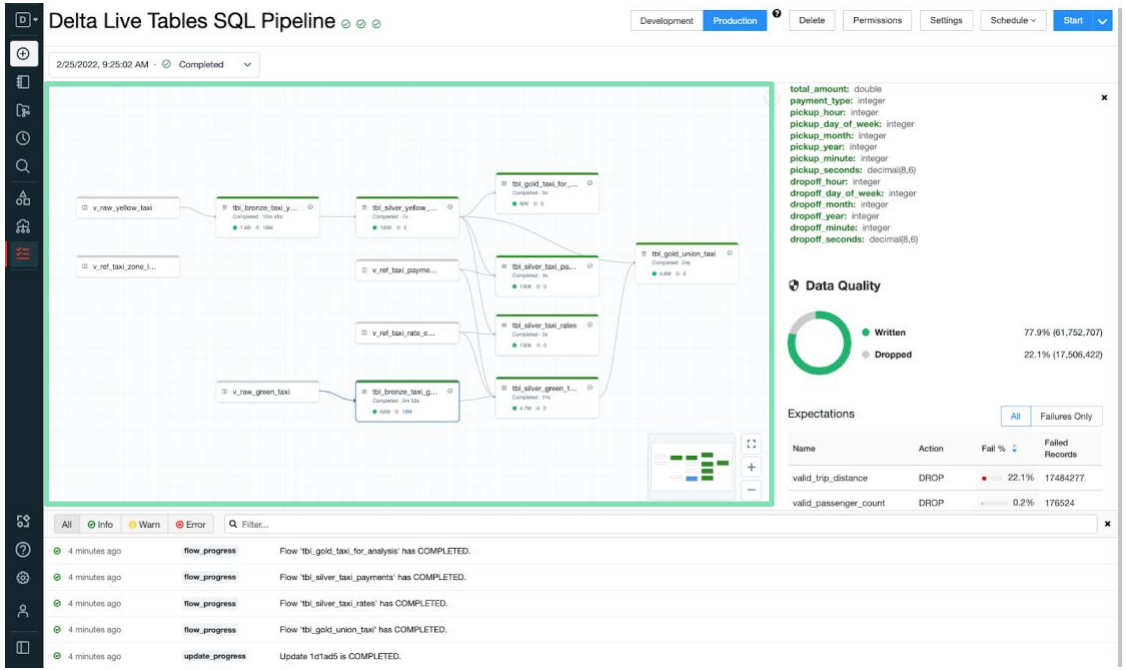
- Track number of bad records
- Drop bad records
- Abort processing for a single bad record

What about operations?

Pipelines UI (1 of 5)

A one stop shop for ETL debugging and operations

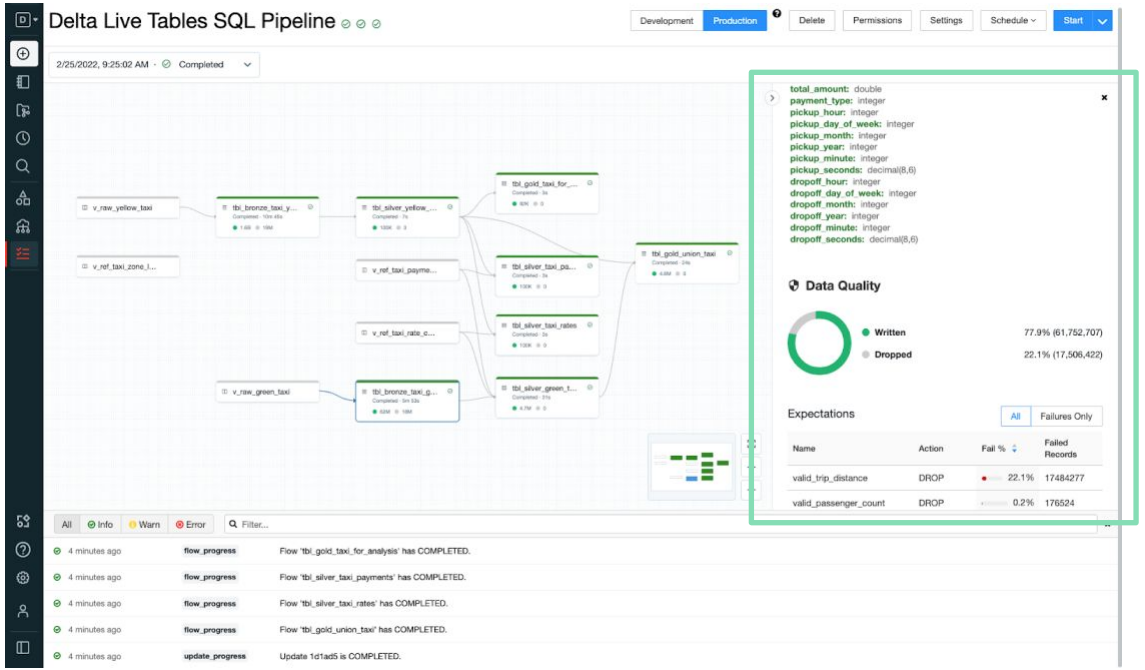
- Visualize data flows between tables



Pipelines UI (2 of 5)

A one stop shop for ETL debugging and operations

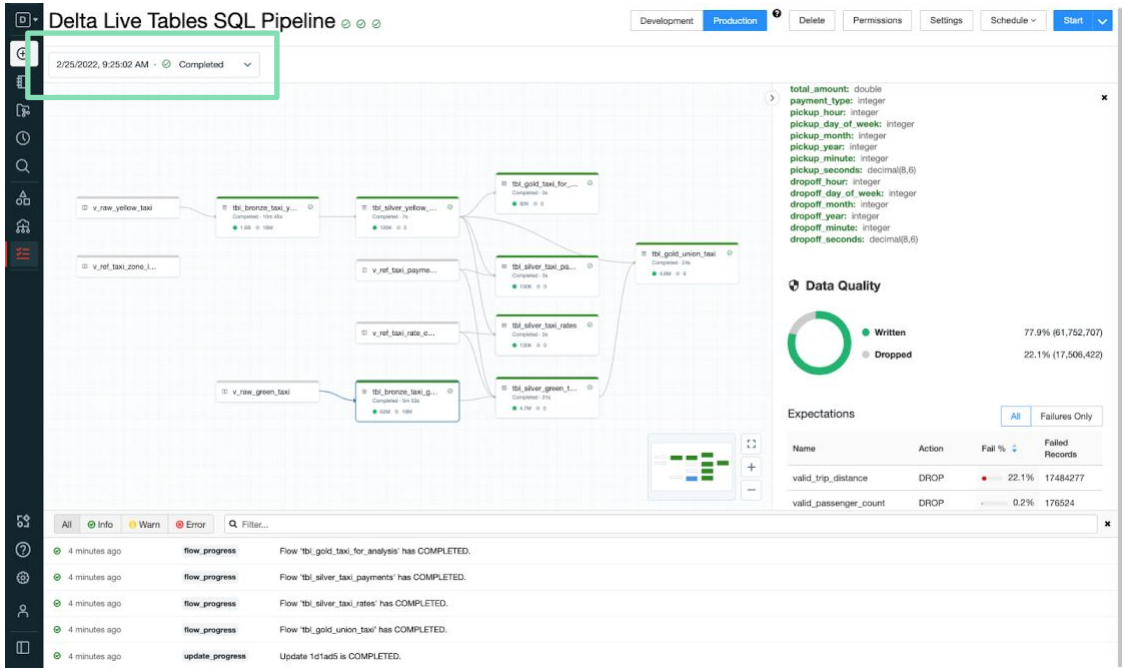
- Visualize data flows between tables
- Discover metadata and quality of each table



Pipelines UI (3 of 5)

A one stop shop for ETL debugging and operations

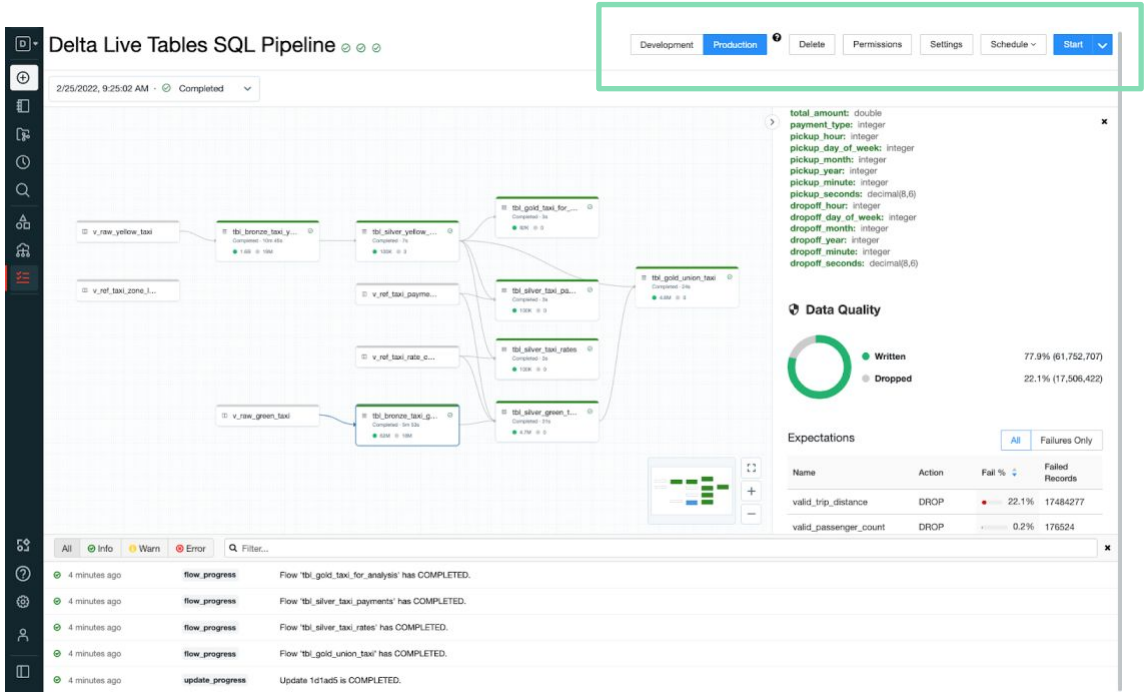
- Visualize data flows between tables
- Discover metadata and quality of each table
- Access to historical updates



Pipelines UI (4 of 5)

A one stop shop for ETL debugging and operations

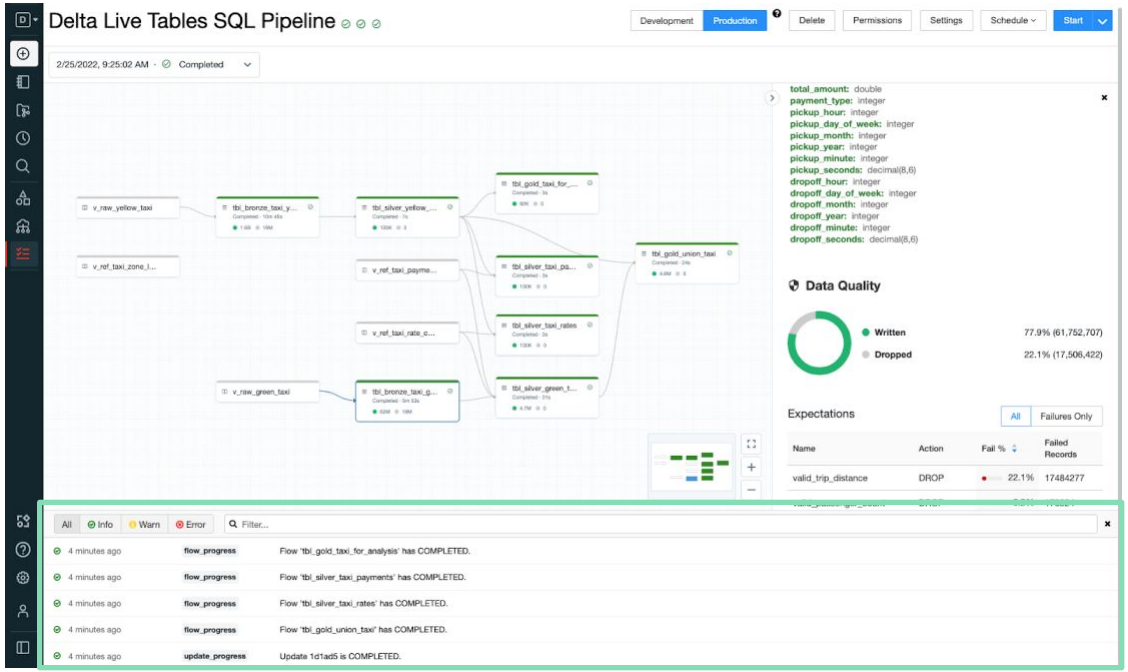
- Visualize data flows between tables
- Discover metadata and quality of each table
- Access to historical updates
- Control operations



Pipelines UI (5 of 5)

A one stop shop for ETL debugging and operations

- Visualize data flows between tables
- Discover metadata and quality of each table
- Access to historical updates
- Control operations
- Dive deep into events



The Event Log

The event log automatically records all pipelines operations.

Operational Statistics	Provenance	Data Quality
Time and current status, for all operations		
Pipeline and cluster configurations		
Row counts	Table schemas, definitions, and declared properties	Expectation pass / failure / drop statistics
	Table-level lineage	Input/Output rows that caused expectation failures
	Query plans used to update tables	

How can I use parameters?


Modularize your code with configuration

Avoid hard coding paths, topic names, and other constants in your code.

A pipeline's configuration is a map of key value pairs that can be used to parameterize your code:

- Improve code readability/ma
- Reuse code in multiple pipeli


```
CREATE STREAMING LIVE TABLE data AS
SELECT * FROM cloud_files("${my_etl.input_path}",
"json")


@dlt.table
def data():
  input_path = spark.conf.get("my_etl.input_path")

spark.readStream.format("cloud_files").load(input_path)
```


How can I do change data capture (CDC)?

APPLY CHANGES INTO for CDC

Maintain an up-to-date replica of a table stored elsewhere

```
APPLY CHANGES INTO LIVE.cities
FROM STREAM(LIVE.city_updates)
KEYS (id)
SEQUENCE BY ts
```

{UPDATE}
{DELETE}
{INSERT}

Up-to-date
Snapshot

APPLY CHANGES INTO for CDC

Maintain an up-to-date replica of a table stored elsewhere

```
APPLY CHANGES INTO LIVE.cities
FROM STREAM(LIVE.city_updates)
KEYS (id)
SEQUENCE BY ts
```

city_updates

```
{"id": 1, "ts": 1, "city": "Bekerly, CA"}
```

A target for the changes to be applied to.

cities

id	city

APPLY CHANGES INTO for CDC

Maintain an up-to-date replica of a table stored elsewhere

```
APPLY CHANGES INTO LIVE.cities
FROM STREAM(LIVE.city_updates)
KEYS (id)
SEQUENCE BY ts
```

A source of changes, currently this has to be a stream.

city_updates

```
{"id": 1, "ts": 1, "city": "Bekerly, CA"}
```


APPLY CHANGES INTO for CDC

Maintain an up-to-date replica of a table stored elsewhere

```
APPLY CHANGES INTO LIVE.cities
FROM STREAM(LIVE.city_updates)
KEYS (id)
SEQUENCE BY ts
```

A unique key that can be used to identify a given row.

city_updates

```
{"id": 1, "ts": 1, "city": "Bekerly, CA"}
```

cities

APPLY CHANGES INTO for CDC

Maintain an up-to-date replica of a table stored elsewhere

```
APPLY CHANGES INTO LIVE.cities
FROM STREAM(LIVE.city_updates)
KEYS (id)
SEQUENCE BY ts
```

city_updates

```
{"id": 1, "ts": 100, "city": "Bekerly, CA"}
```

A sequence that can be used to order changes:

- Log sequence number (lsn)
- Timestamp
- Ingestion time

cities

APPLY CHANGES INTO for CDC

Maintain an up-to-date replica of a table stored elsewhere

```
APPLY CHANGES INTO LIVE.cities
FROM STREAM(LIVE.city_updates)
KEYS (id)
SEQUENCE BY ts
```

city_updates

```
{"id": 1, "ts": 100, "city": "Bekerly, CA"}
{"id": 1, "ts": 200, "city": "Berkeley, CA"}
```

cities

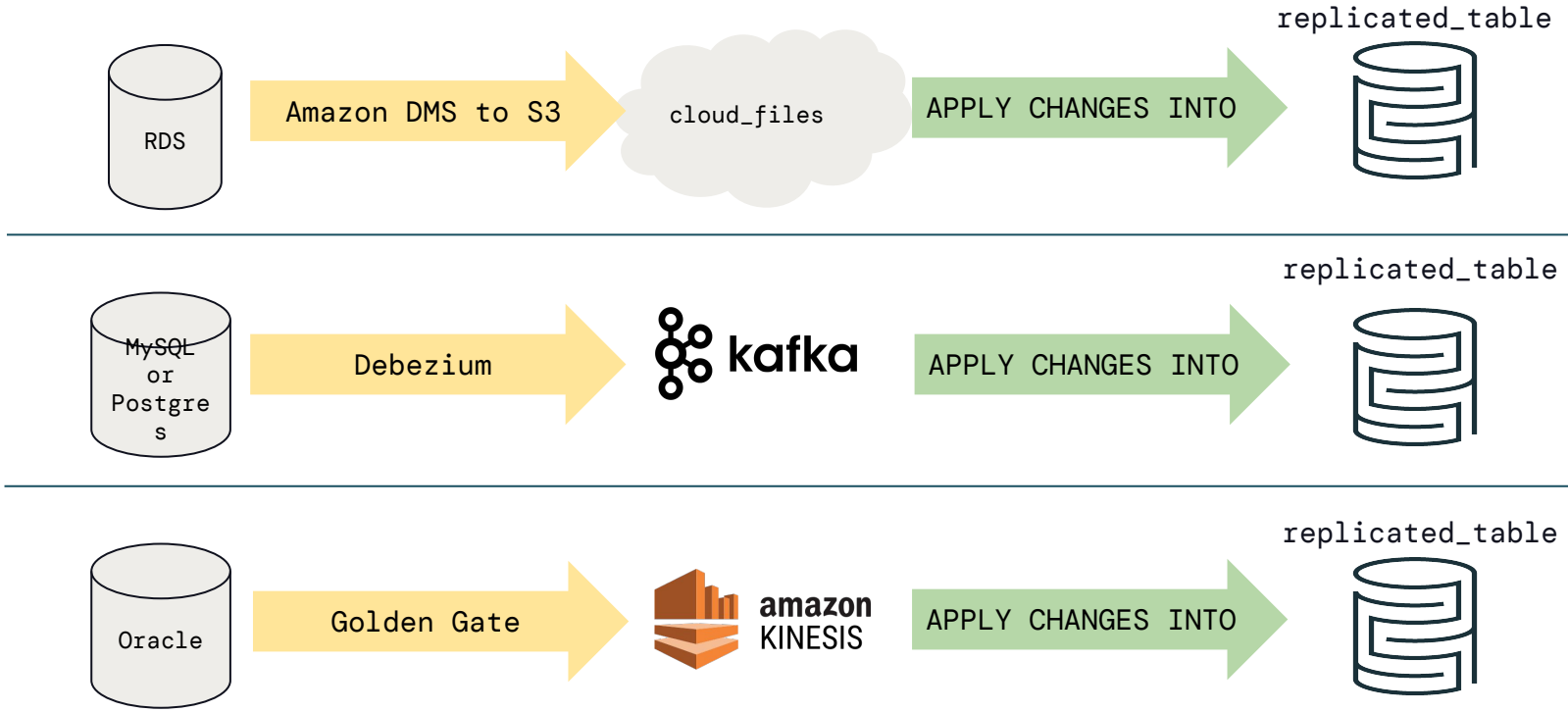
id	city
1	~~Bekerly, CA~~

Berkeley, CA

REFERENCE ARCHITECTURE

Change Data Capture (CDC) from RDBMS

A variety of 3rd party tools can provide a streaming change feed



What do I *no longer* need to manage with DLT?

Automated Data Management

DLT automatically optimizes data for performance & ease-of-use

Best Practices

What:

DLT encodes Delta best practices automatically when creating DLT tables.

How:

DLT sets the following properties:

- `optimizeWrite`
- `autoCompact`
- `tuneFileSizesForRewrites`

Physical Data

What:

DLT automatically manages your physical data to minimize cost and optimize performance.

How:

- runs vacuum daily
- runs optimize daily

You still can tell us how you want it organized (ie ZORDER)

Schema Evolution

What:

Schema evolution is handled for you

How:

Modifying a live table transformation to add/remove/rename a column will automatically do the right thing.

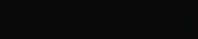
When removing a column in a streaming live table, old values are preserved.

Demo:
DE 4.1 - Using the Delta Live Tables UI

Demo:
DE 4.1.1 – Fundamentals of DLT Syntax

Demo:
DE 4.1.2 – More DLT SQL Syntax

Demo:
DE 4.2 – Delta Live Tables: Python vs SQL

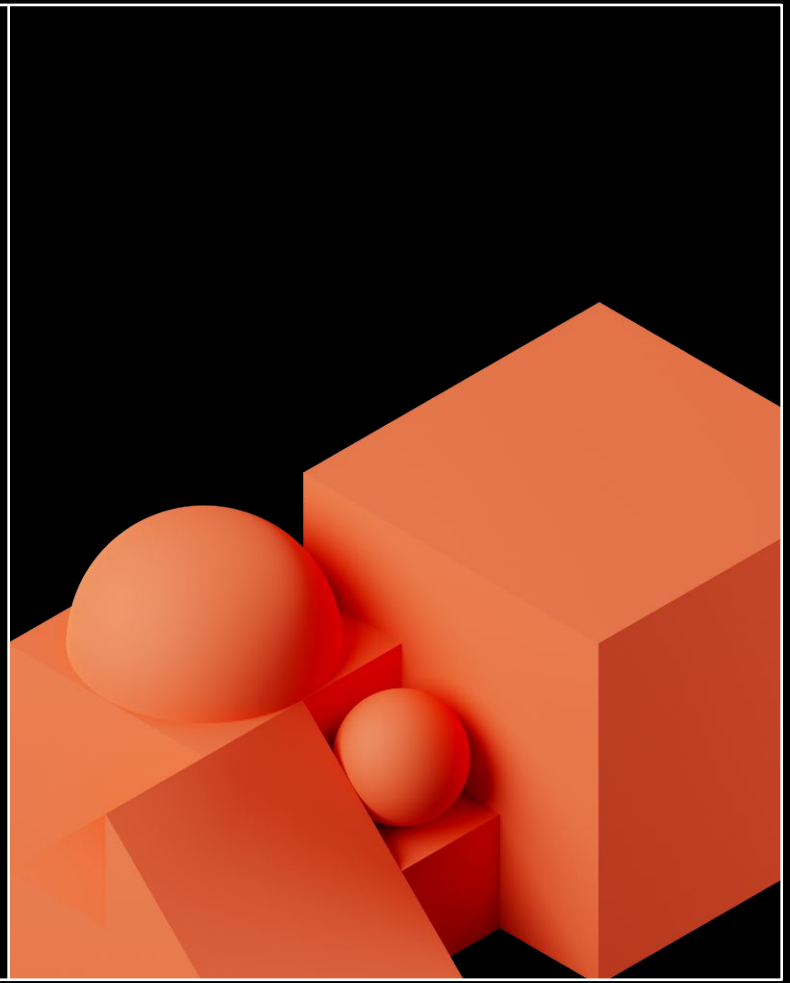


Demo:
DE 4.3 – Exploring the Results of a DLT Pipeline

Demo:
DE 4.4 – Exploring the Pipeline Events Logs

Lab:
DE 4.1.3 - Troubleshooting DLT Syntax Lab

Deploy Workloads with Databricks Workflows



Module Agenda

Deploy Workloads with Databricks Workflows

Introduction to Workflows

Building and Monitoring Workflow Jobs

DE 5.1 - Scheduling Tasks with the Jobs UI

DE 5.2L - Jobs Lab

Introduction to Workflows

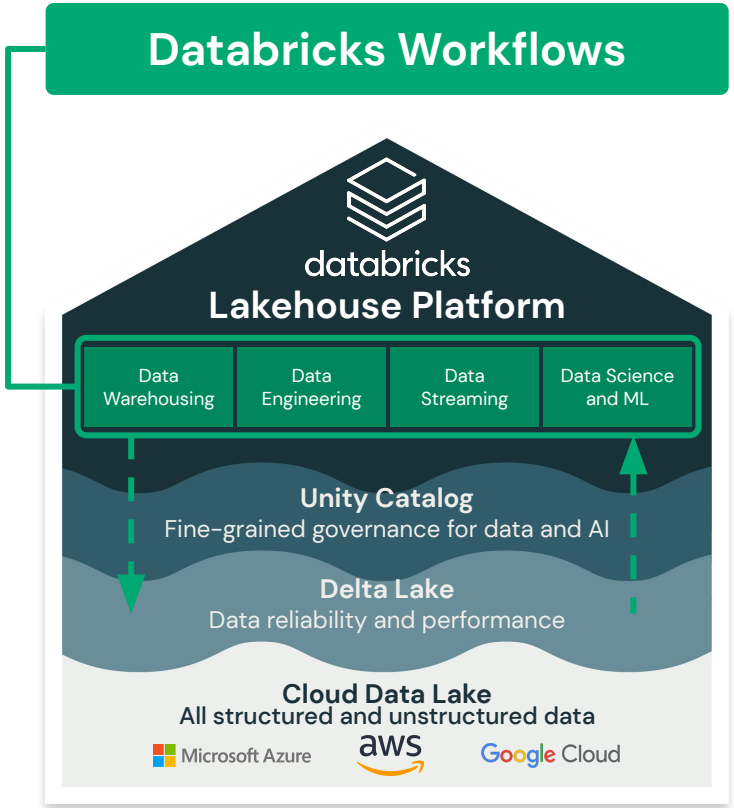
Lesson Objectives

1. Describe the main features and use cases of Databricks Workflows
2. Create a task orchestration workflow composed of various task types
3. Utilize monitoring and debugging features of Databricks Workflows
4. Describe workflow best practices

Databricks Workflows

Workflows is a **fully-managed cloud-based general-purpose task orchestration service** for the entire Lakehouse.

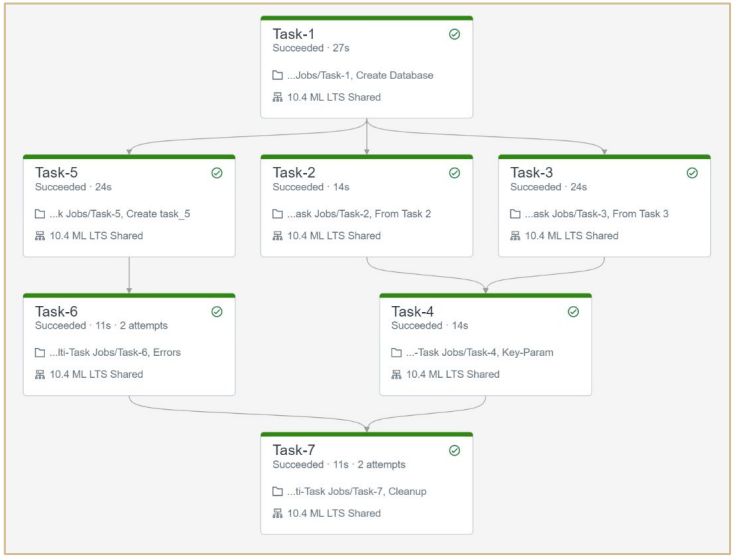
Workflows is a service for data engineers, data scientists and analysts to build reliable data, analytics and AI workflows on any cloud.



Databricks Workflows

Databricks has two main task orchestration services:

- **Workflow Jobs (Workflows)**
  - Workflows for every job
- **Delta Live Tables (DLT)**
  - Automated data pipelines for Delta Lake



Note: DLT pipeline can be a task in a workflow

DLT versus Jobs

Considerations

	Delta Live Tables	Workflow Jobs
Source	Notebooks only	JARs, notebooks, DLT, application written in Scala, Java, Python
Dependencies	Automatically determined	Manually set
Cluster	Self-provisioned	Self-provisioned or existing
Timeouts and Retries	Not supported	Supported
Import Libraries	Not supported	Supported

DLT versus Jobs

Use Cases

Orchestration of Dependent Jobs

Jobs running on schedule, containing dependent tasks/steps

Jobs Workflows

Machine Learning Tasks

Run MLflow notebook task in a job

Jobs Workflows

Arbitrary Code, External API Calls, Custom Tasks

Run tasks in a job which can contain Jar file, Spark Submit, Python Script, SQL task, dbt

Jobs Workflows

Data Ingestion and Transformation

ETL jobs, Support for batch and streaming, Built in data quality constraints, monitoring & logging

Delta Live Tables

Workflows Features

Part 1 of 2

Orchestrate Anything Anywhere

Run diverse workloads for the full data and AI lifecycle, on any cloud. Orchestrate;

- Notebooks
- Delta Live Tables
- Jobs for SQL
- ML models, and more

Fully Managed

Remove operational overhead with a fully managed orchestration service enabling you to focus on your workflows not on managing your infrastructure

Simple Workflow Authoring

An easy point-and-click authoring experience for all your data teams not just those with specialized skills

Workflows Features

Part 2 of 2

Deep Platform Integration

Designed and built into your lakehouse platform giving you deep monitoring capabilities and centralized observability across all your workflows

Proven Reliability

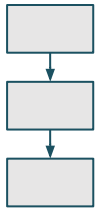
Have full confidence in your workflows leveraging our proven experience running tens of millions of production workloads daily across AWS, Azure, and GCP

How to Leverage Workflows

- Allows you to build simple ETL/ML task orchestration
- Reduces infrastructure overhead
- Easily integrate with external tools
- Enables non-engineers to build their own workflows using simple UI
- Cloud-provider independent
- Enables re-using clusters to reduce cost and startup time

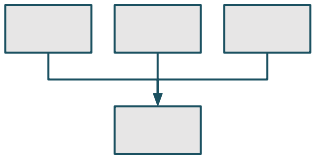
Common Workflow Patterns

Sequence



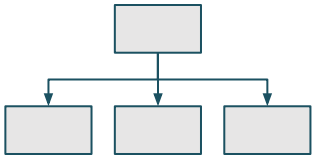
Sequence
- Data transformation/ processing/cleaning
- Bronze/silver/gold tables

Funnel



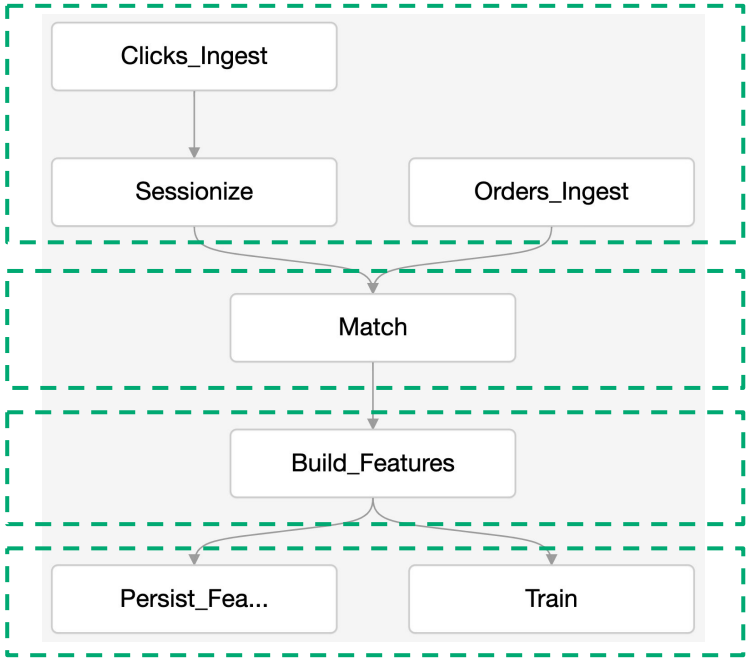
Funnel
- Multiple data sources
- Data collection

Fan-out



Fan-out, star pattern
- Single data source
- Data ingestion and distribution

Example Workflow



Data ingestion funnel
E.g. Auto Loader, DLT

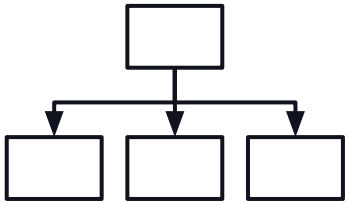
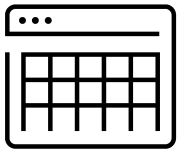
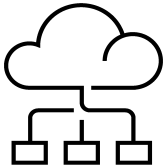
Data filtering, quality assurance, transformation
E.g. DLT, SQL, Python

ML feature extraction
E.g. MLflow

Persisting features and training prediction model

Building and Monitoring Workflow Jobs

Workflows Job Components

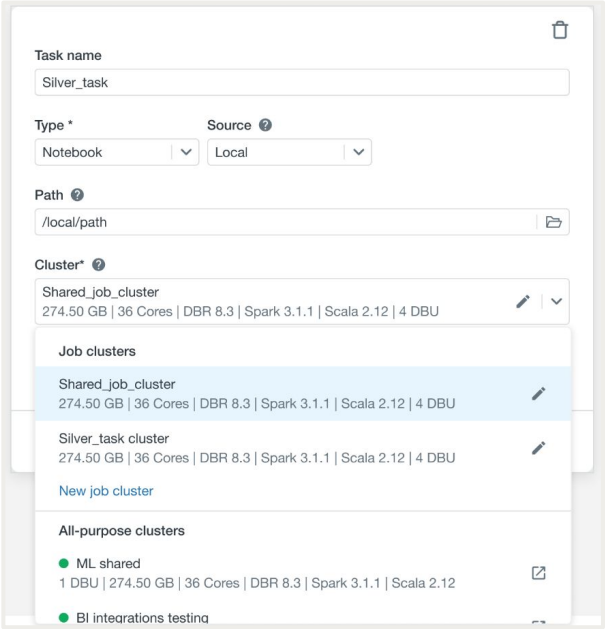
TASKS	SCHEDULE	CLUSTER
		
What?	When?	How?

Creating a Workflow

Task Definition

While creating a task;

- Define the task type
- Choose the cluster type
  - Job clusters and All-purpose clusters can be used.
  - A cluster can be used by multiple tasks. This reduces cost and startup time.
- If you want to create a new cluster, you must have required permissions.
- Define task dependency if task depends on another task

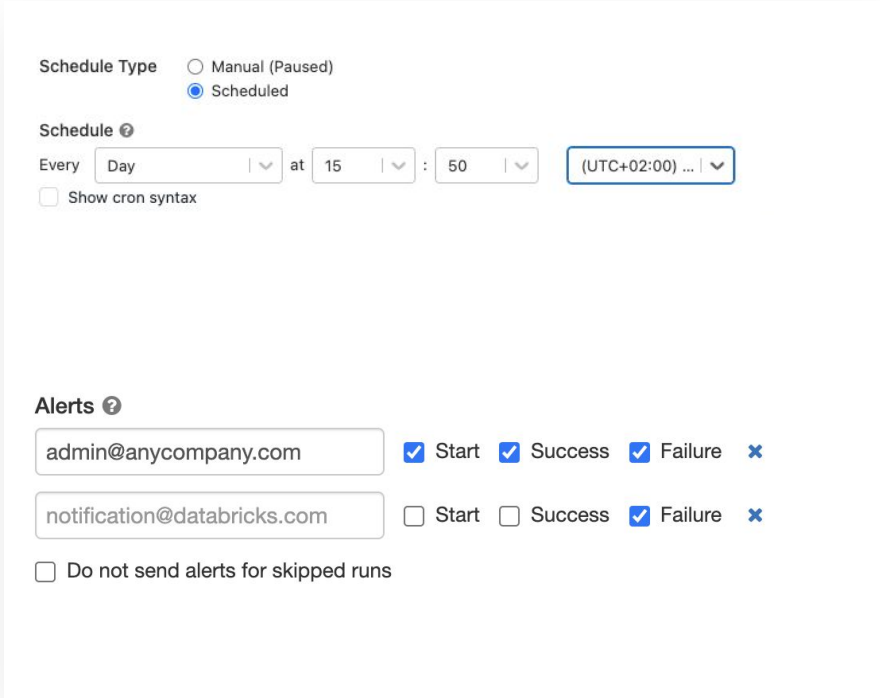


Monitoring and Debugging

Scheduling and Alerts

You can run your jobs **immediately** or **periodically** through an easy-to-use scheduling system.

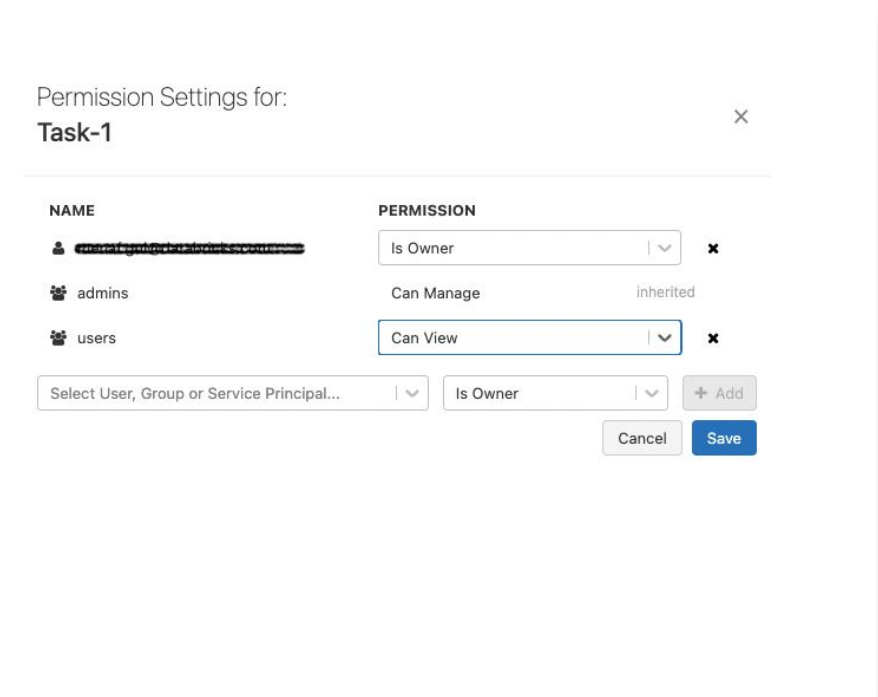
You can specific alerts to be notified when runs of a job **begin, complete or fail**. Notifications can be sent via email, Slack or AWS SNS.



Monitoring and Debugging

Access Control

Workflows integrates with existing resources access controls, enabling you to easily manage access across different teams.

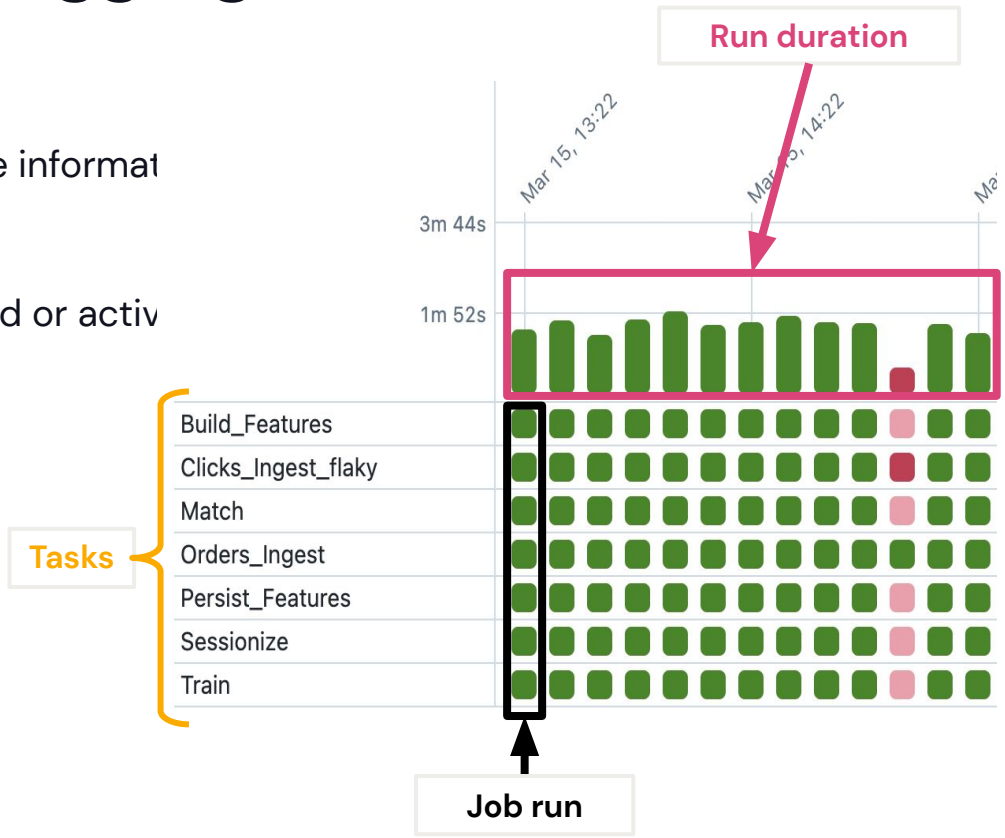


Monitoring and Debugging

Job Run History

Workflows keeps track of job runs and save informat
in the job run.

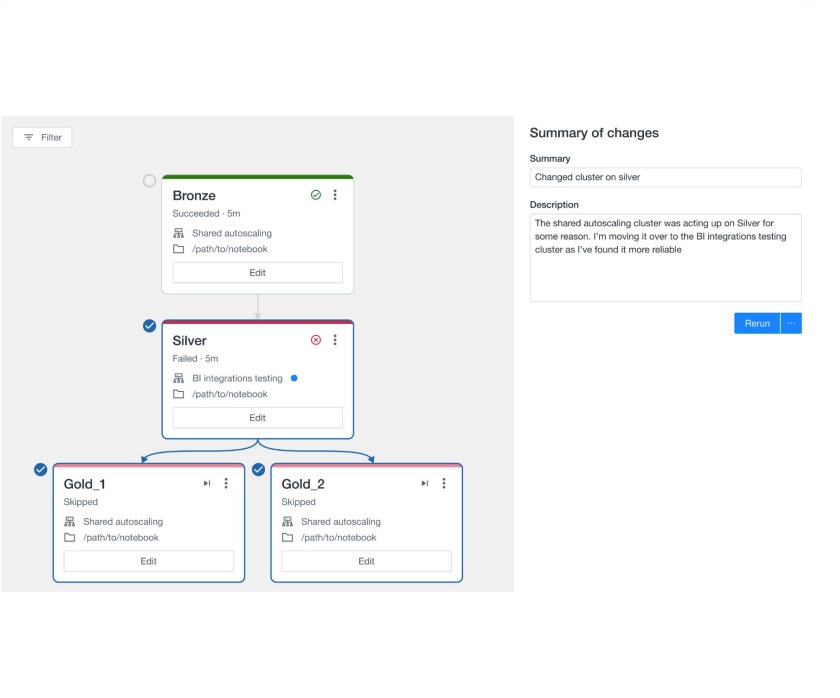
Navigate to the Runs tab to view completed or activ



Monitoring and Debugging

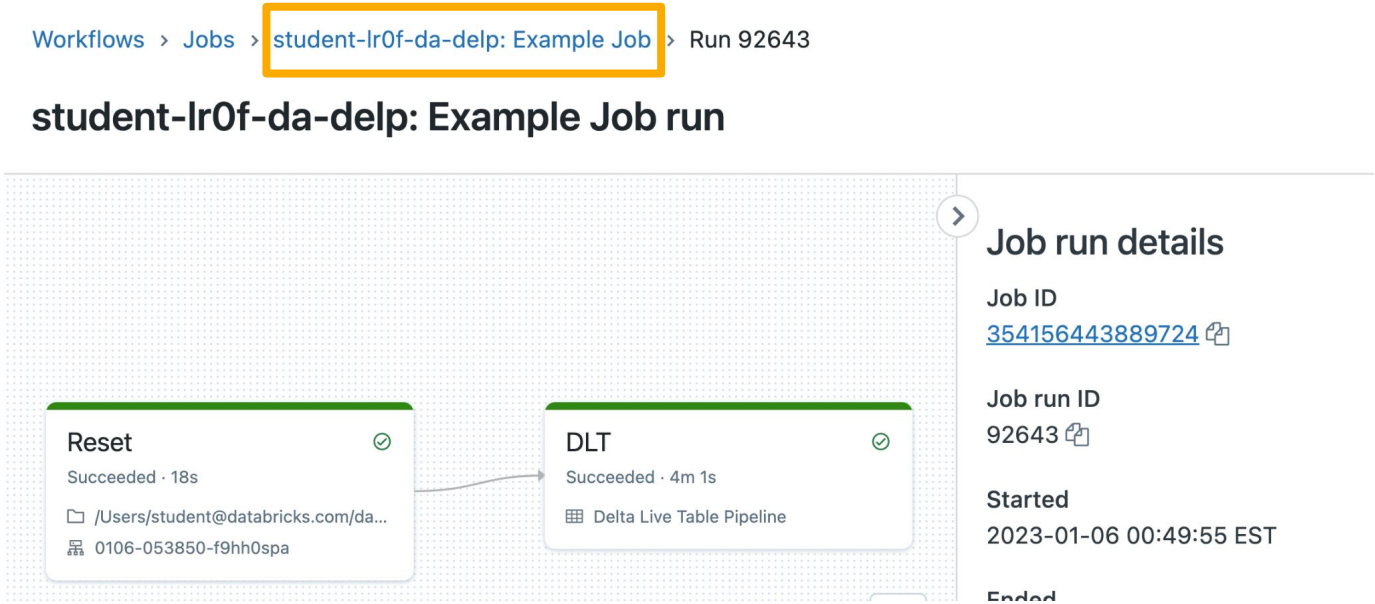
Repair a Failed Job Run

Repair feature allows you to re-run only the failed task and sub-tasks, which reduces the time and resources required to recover from unsuccessful job runs.



Navigating the Jobs UI

Use breadcrumbs to navigate back to your job from a specific run page

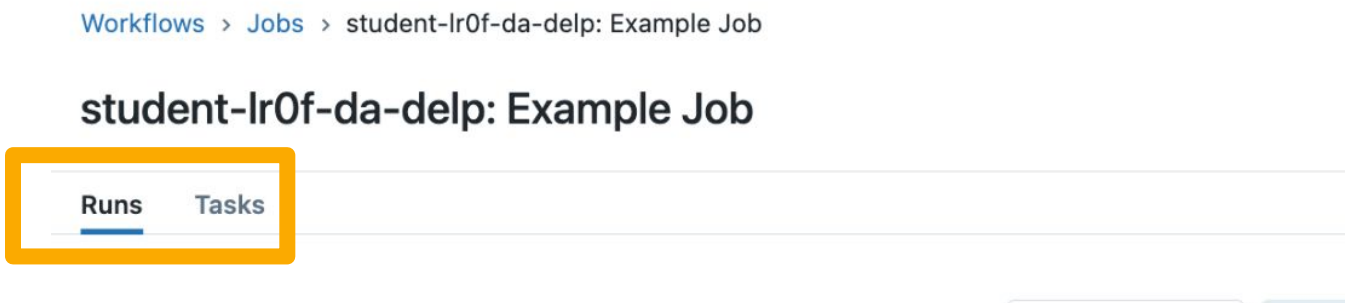


Navigating the Jobs UI

Runs vs Tasks tabs on the job page

Use **Runs** tab to view completed or active runs for the job

Use **Tasks** tab to modify or add tasks to the job

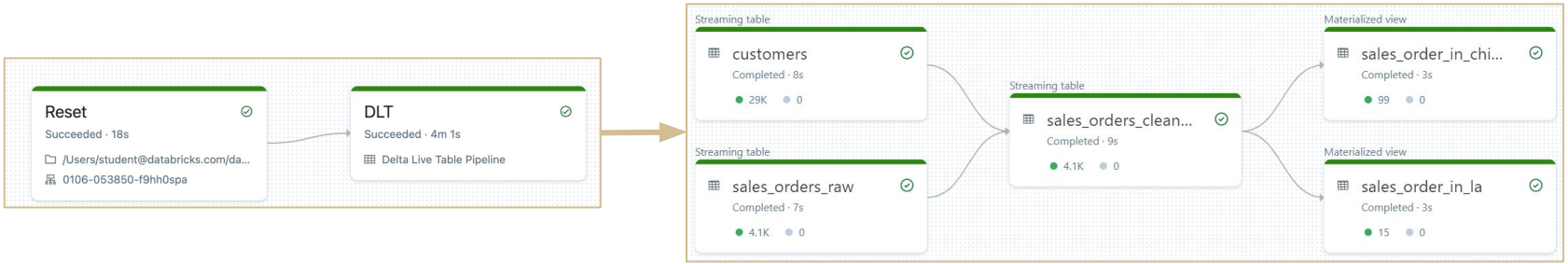


Demo:
DE 5.1.1: Task Orchestration

Demo: Task Orchestration

DE 5.1.1 – Task Orchestration

- Schedule a notebook task in a Databricks Workflow Job
- Describe job scheduling options and differences between cluster types
- Review Job Runs to track progress and see results
- Schedule a DLT pipeline task in a Databricks Workflow Job
- Configure dependency between tasks via Databricks Workflows UI

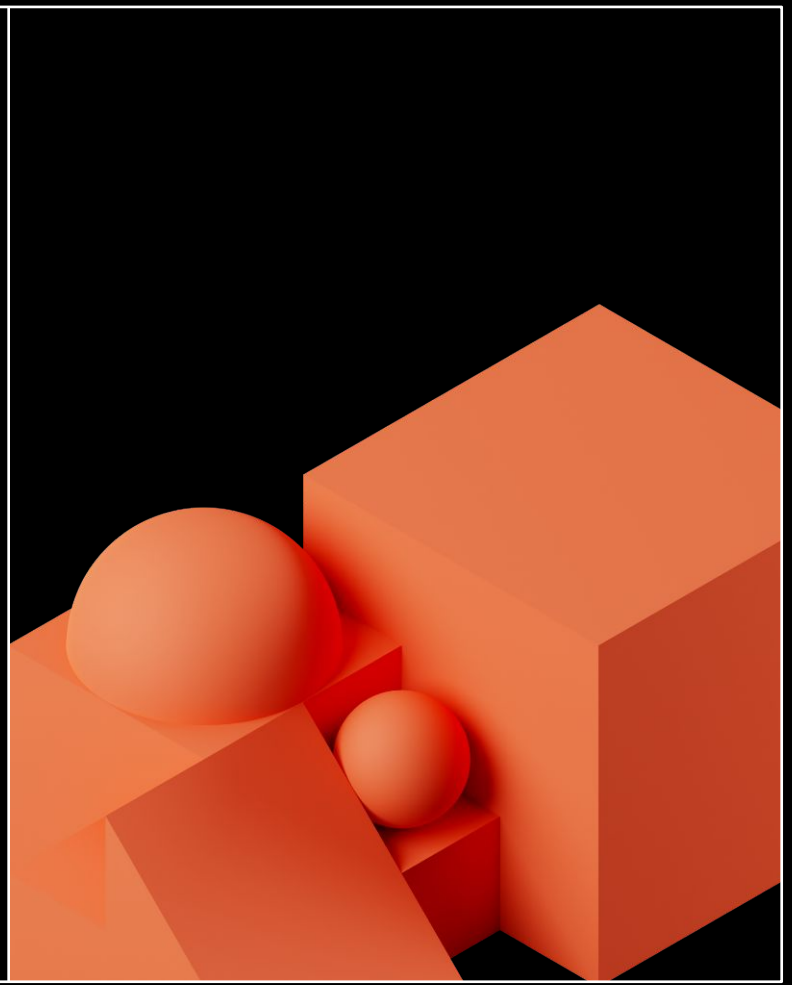


Lab:
DE 5.2.1.L: Task Orchestration

Lab: Task Orchestration

DE 5.2.1.L – Task Orchestration

Manage Data Access for Analytics



Lesson Objectives

By the end of this course, you will be able to:

1. Describe Unity Catalog key concepts and how it integrates with the Databricks platform
2. Access Unity Catalog through clusters and SQL warehouses
3. Create and govern data assets in Unity Catalog
4. Adopt Databricks recommendations into your organization's Unity Catalog-based solutions

Module Agenda

Manage Data Access for Analytics with Unity Catalog

Introduction to Unity Catalog

DE 6.1 - Introduction to Unity Catalog

DE 6.2 - Overview of Data Governance

DE 6.3 - Unity Catalog Key Concepts

DE 6.4 - Unity Catalog Architecture

DE 6.5 - Unity Catalog Identities

DE 6.6 - Managing Principals in Unity Catalog

DE 6.7 - Managing Catalog Metastores

Compute Resources in Unity Catalog

DE 6.8 - Compute Resources

DE 6.9 - Creating Compute Resources

Data Access Control in Unity Catalog

DE 6.10 - Data Access Control in Databricks

DE 6.11 - Security Model

DE 6.12 - External Storage

DE 6.13 - Creating and Governing Data

DE 6.14 - Create and Share Tables

DE 6.15 - Create External Tables

Unity Catalog Best Practices

DE 6.16 - Best Practices

DE 6.17 - Data Segregation

DE 6.18 - Identity Management

DE 6.19 - External Storage

DE 6.20 - Upgrade a Table to Unity Catalog

DE 6.21 - Create Views and Limiting Table Access

Introduction to Unity Catalog

Overview of Data Governance

80% of organizations seeking to scale digital business will fail because they do not take a modern approach to data and analytics governance

Source: Gartner

Data Governance

Four key functional areas

Data Access Control

Control who has access to which data

Data Access Audit

Capture and record all access to data

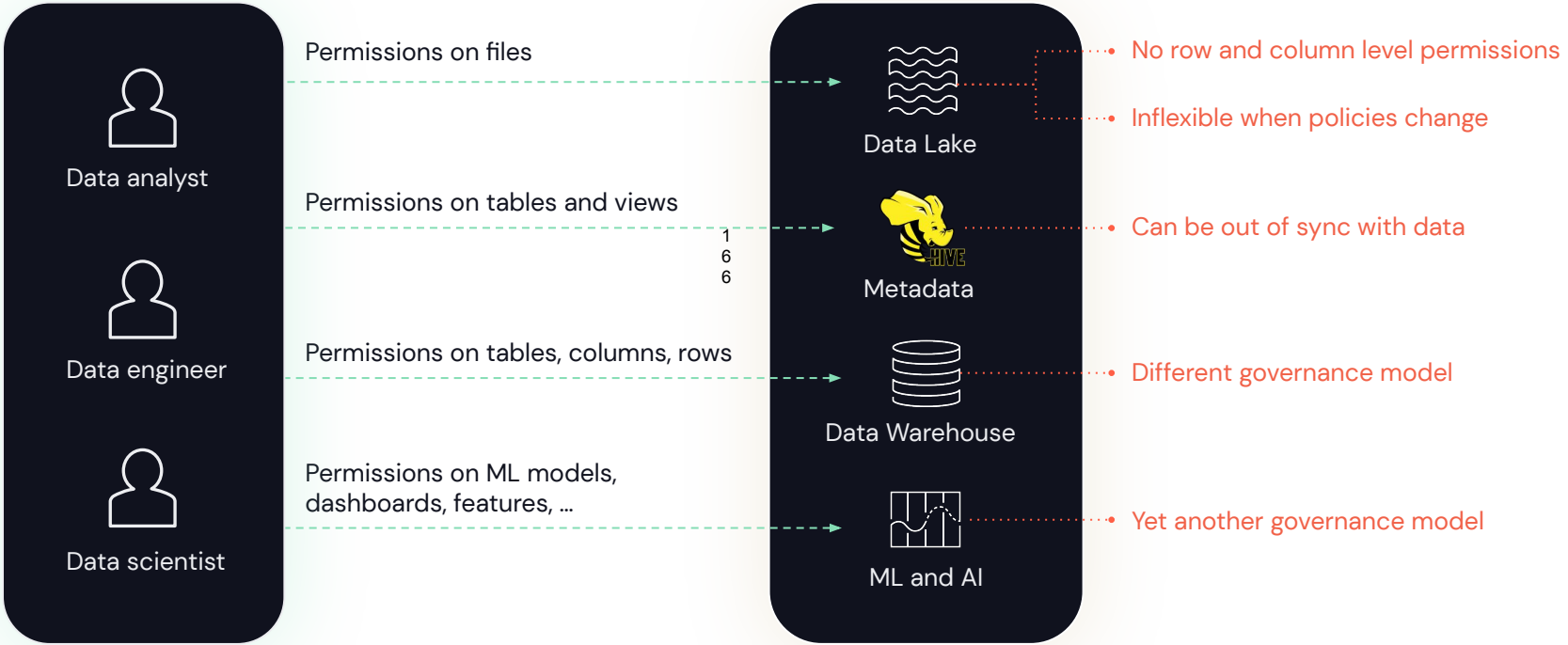
Data Lineage

Capture upstream sources and downstream consumers

Data Discovery

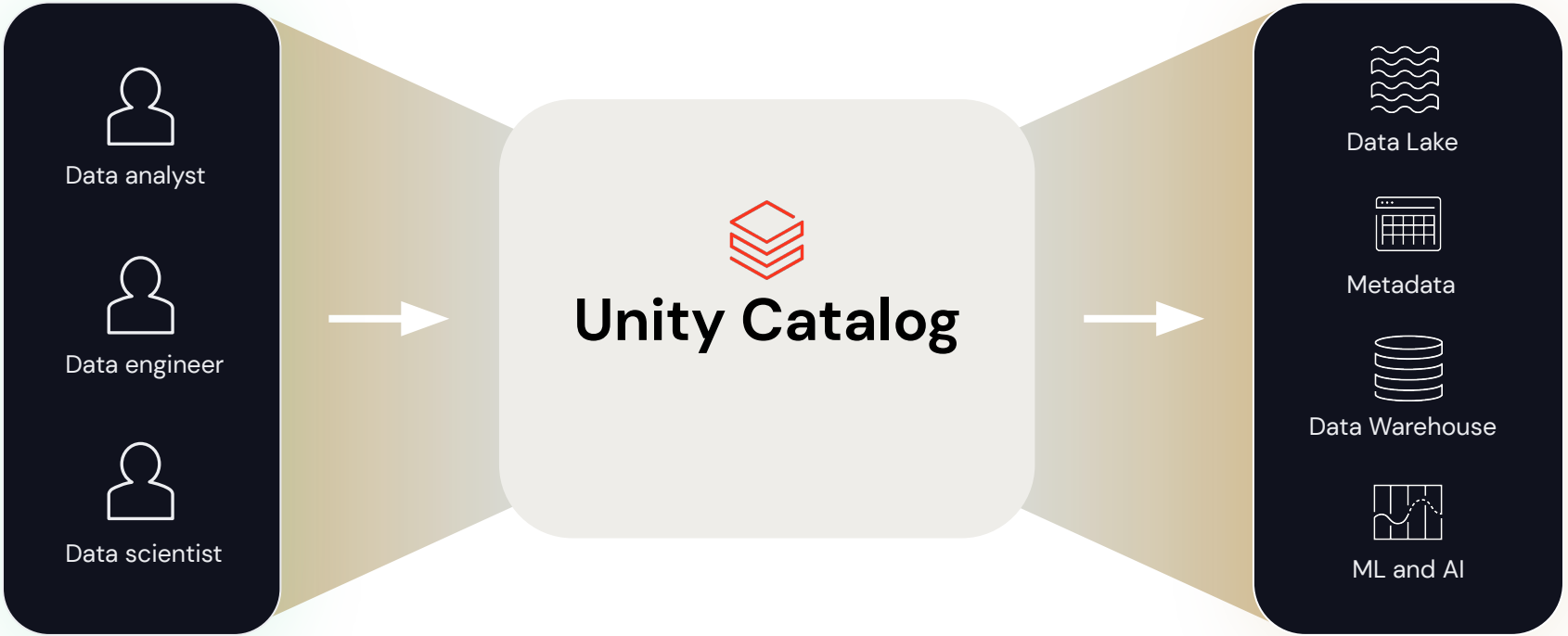
Ability to search for and discover authorized assets

Governance for data, analytics and AI is complex



Databricks Unity Catalog

Unified governance for data, analytics and AI



Unity Catalog

Overview

Unified governance across clouds

Fine-grained governance for data lakes across clouds - based on open standard ANSI SQL.

1

Unified data and AI assets

Centrally share, audit, secure and manage all data types with one simple interface.

2

Unified existing catalogs

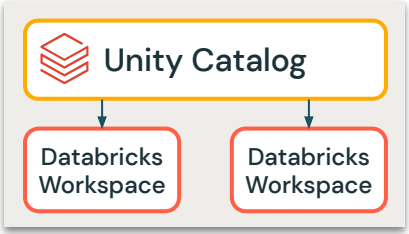
Works in concert with existing data, storage, and catalogs - no hard migration required.

3

Unity Catalog

Key Capabilities

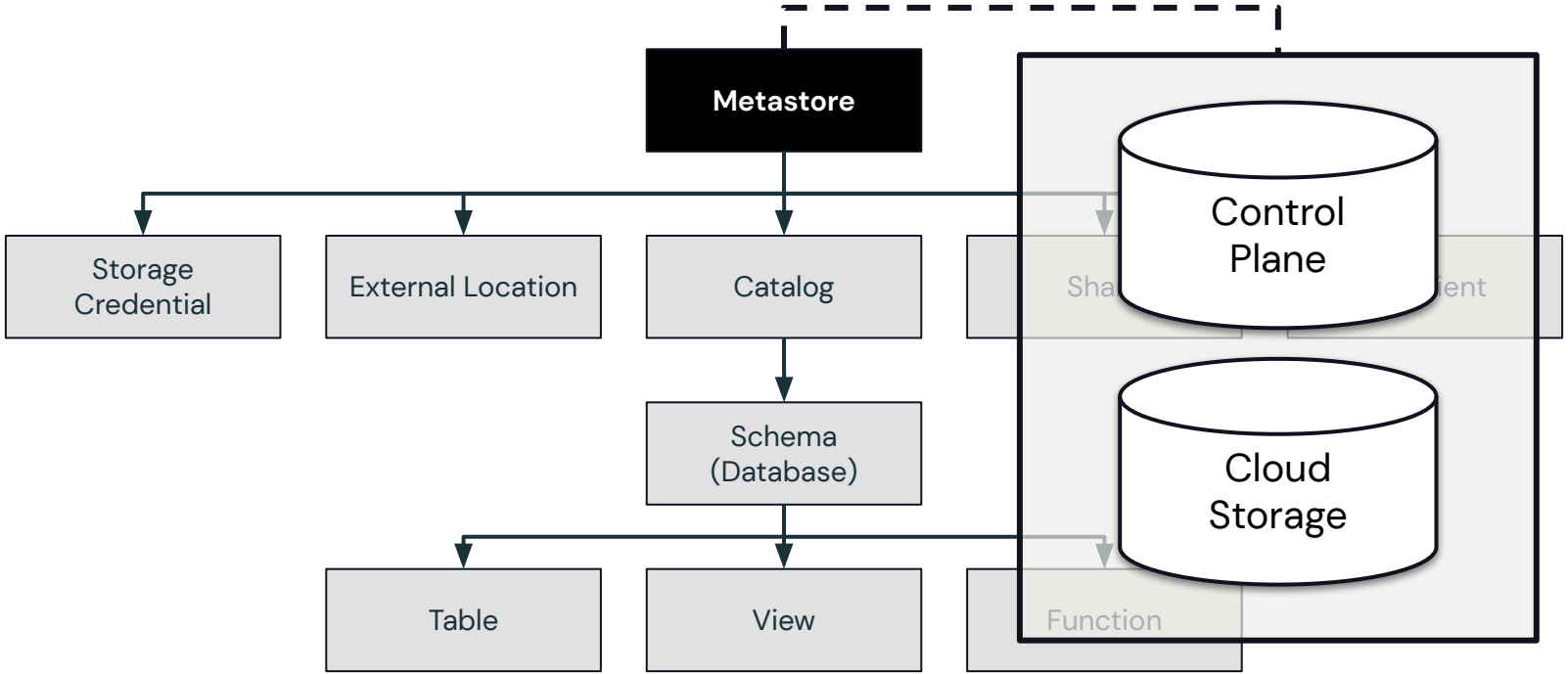
- Centralized metadata and user management
- Centralized data access controls
- Data access auditing
- Data lineage
- Data search and discovery
- Secure data sharing with Delta Sharing



Unity Catalog Key Concepts

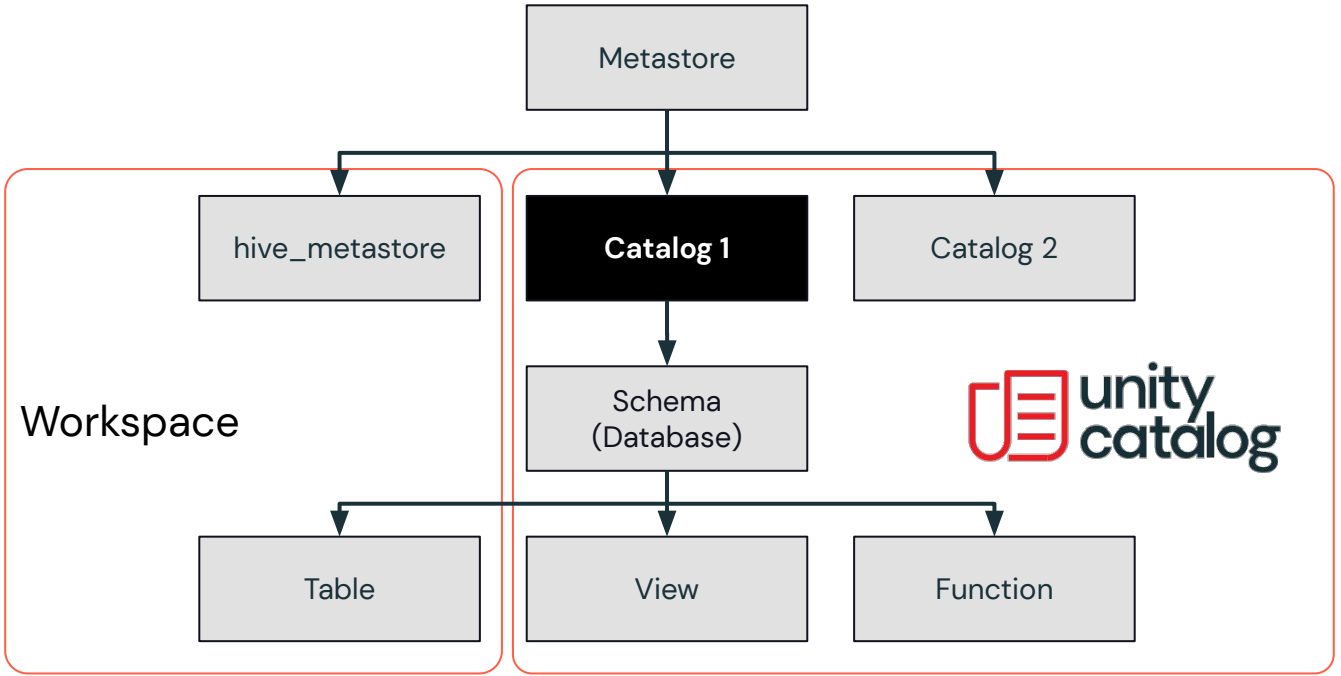
Metastore

Unity Catalog metastore elements



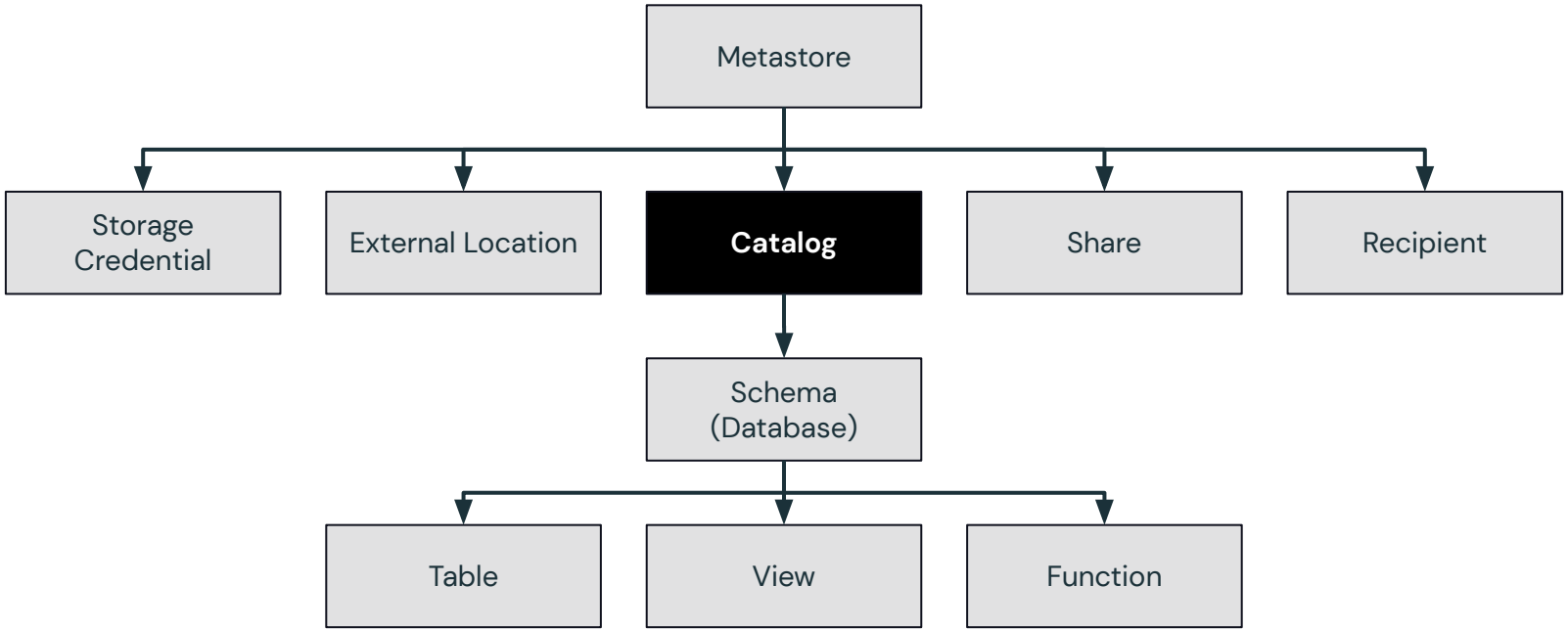
Metastore

Accessing legacy Hive metastore



Catalog

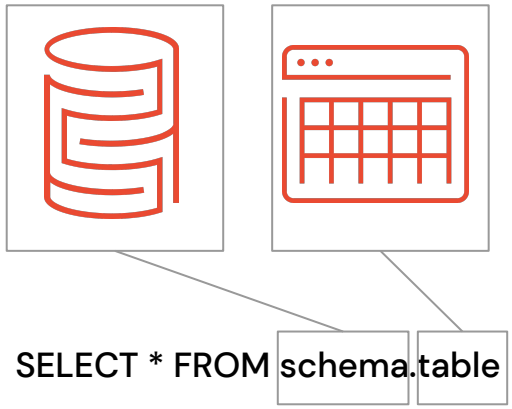
Top-level container for data objects



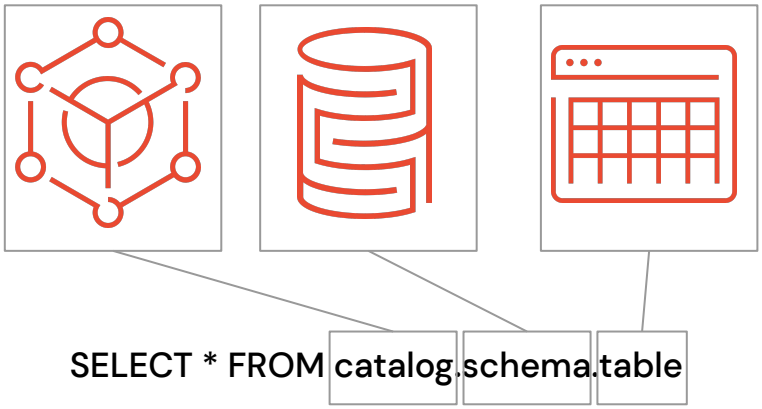
Catalog

Three-level namespace

Traditional SQL two-level namespace

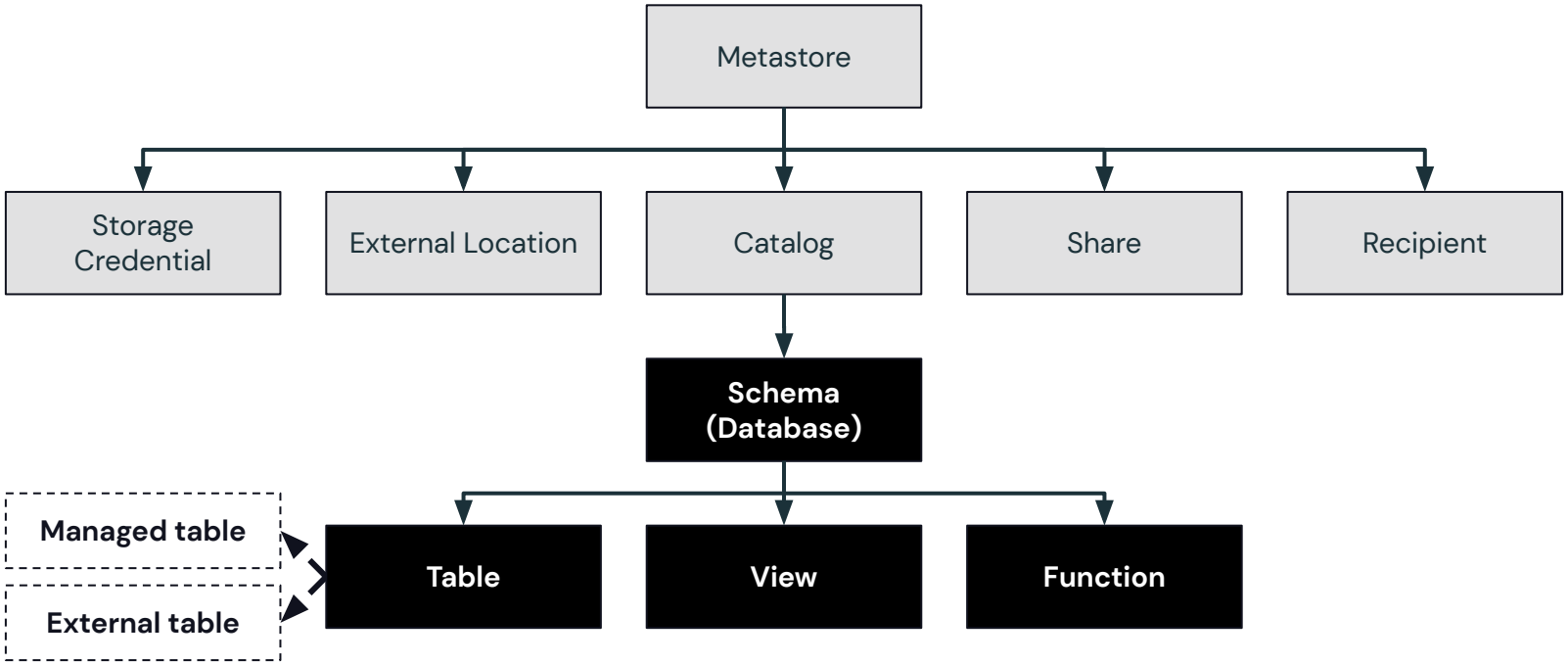


Unity Catalog three-level namespace



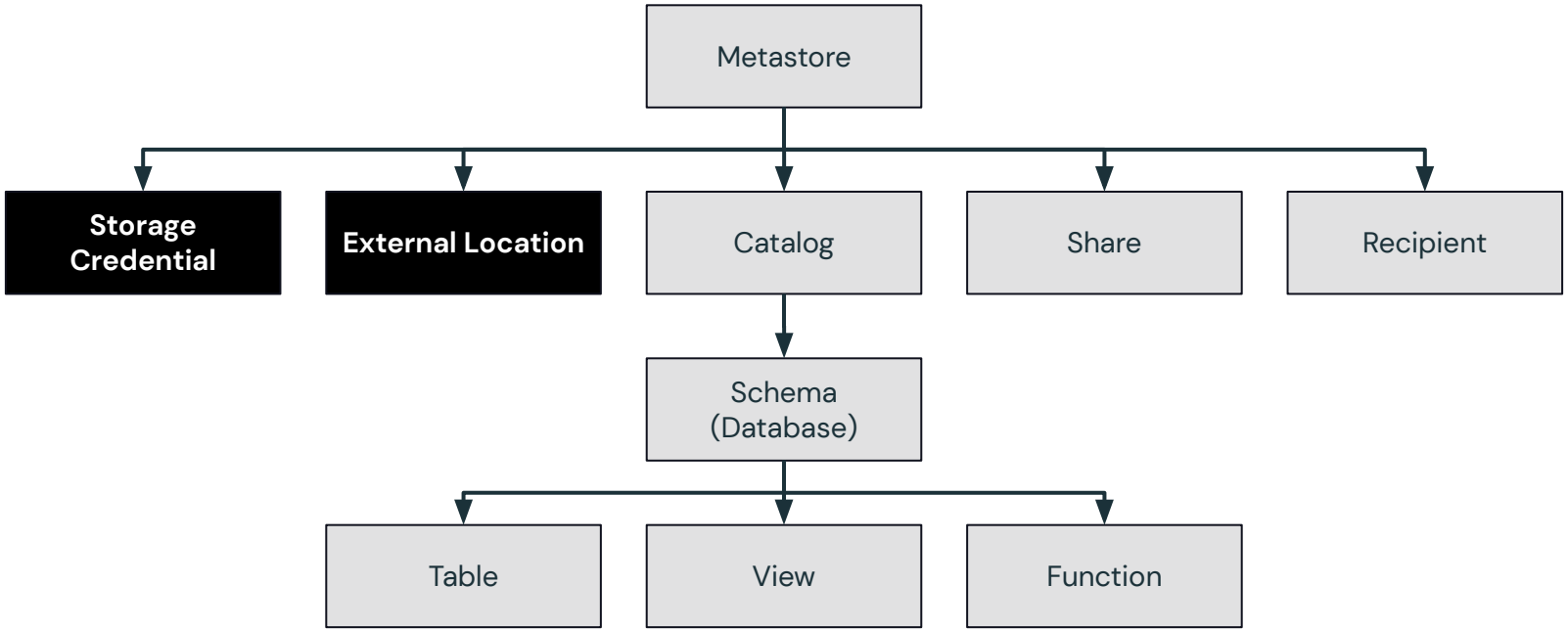
Data Objects

Schema (database), tables, views, functions



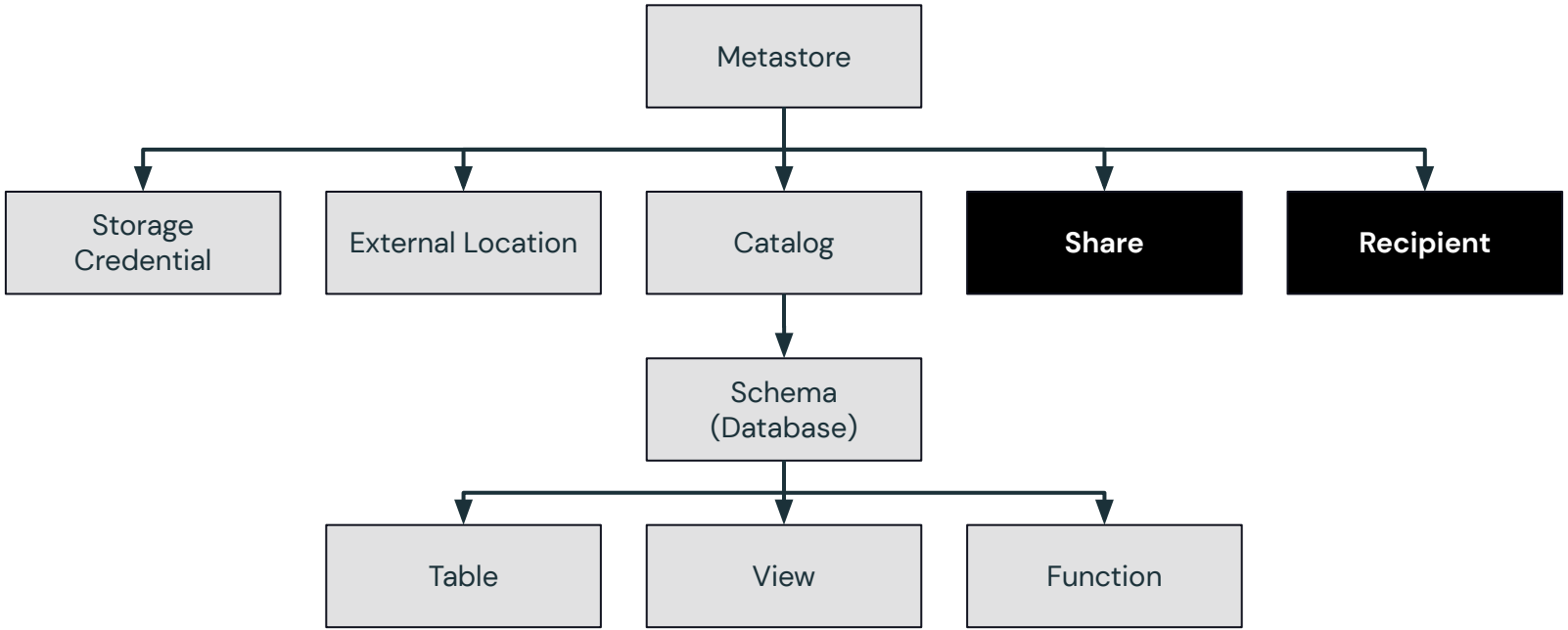
External Storage

Storage credentials and external locations



Delta Sharing

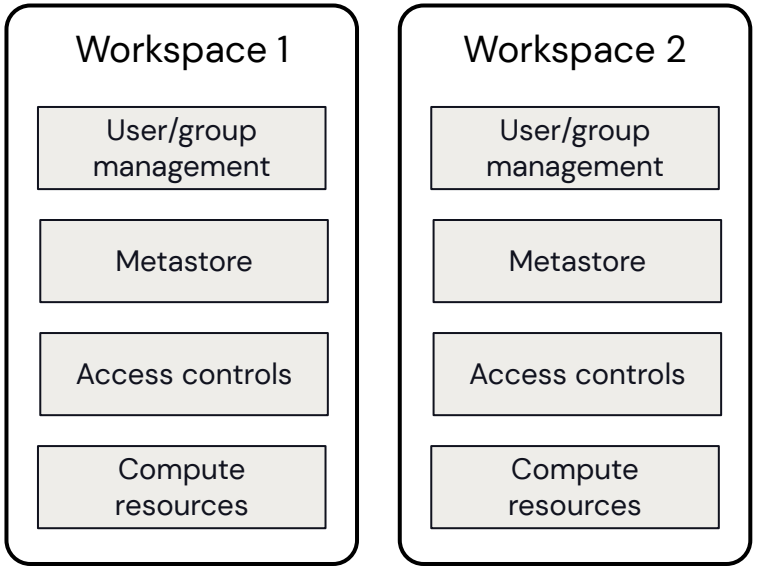
Shares and recipients



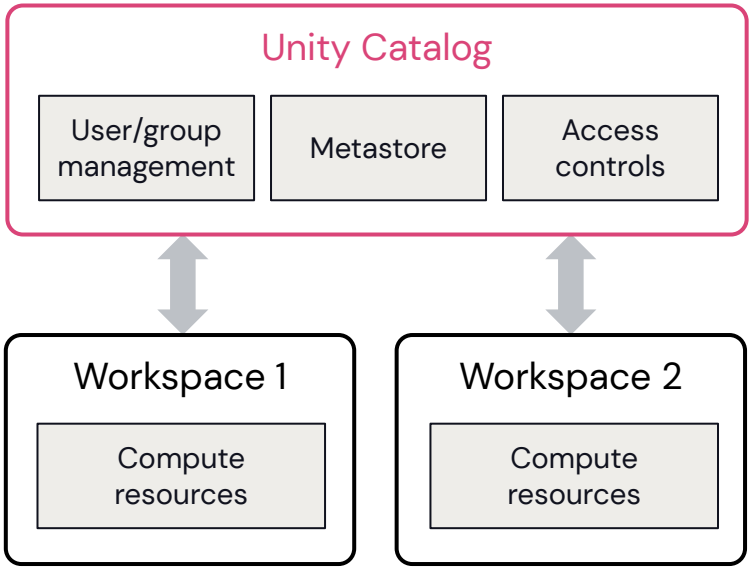
Unity Catalog Architecture

Architecture

Before Unity Catalog

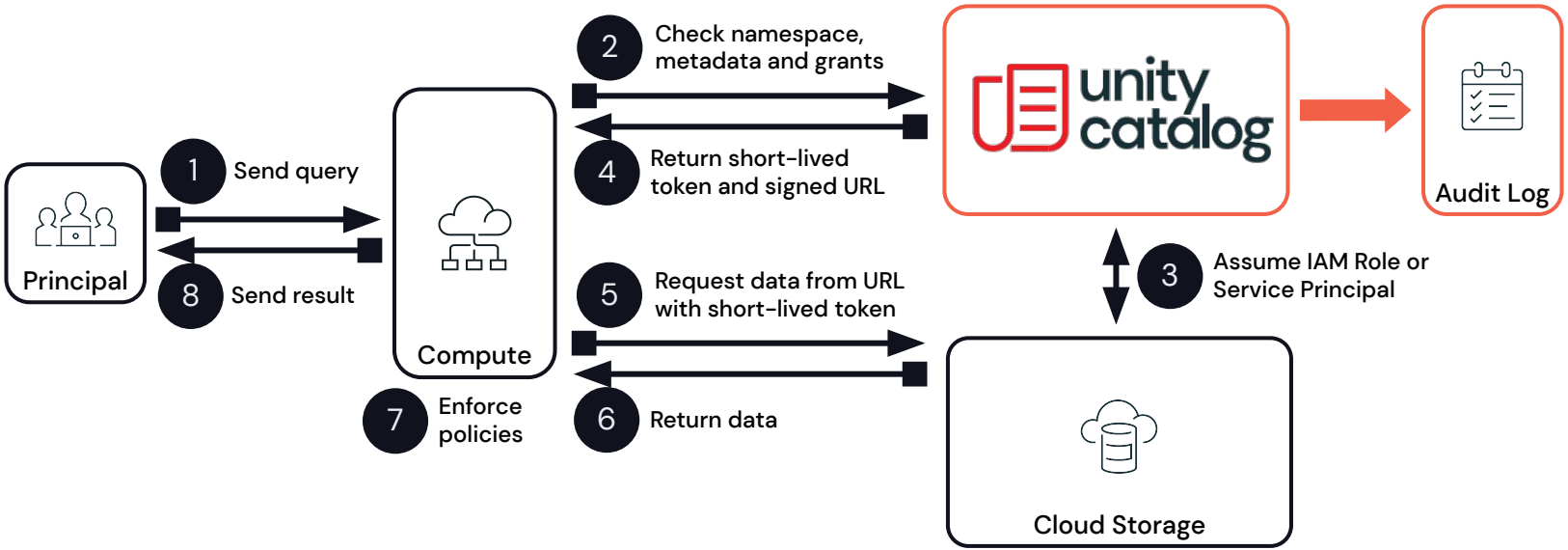


With Unity Catalog



Query Lifecycle

Unity Catalog Security Model



Compute Resources and Unity Catalog

Compute Resources for Unity Catalog

Modes supporting UC

Modes not supporting UC

Cluster Access Mode

No isolation shared

Multiple language support

Single user

Multiple language support, not shareable

Shared

Shareable, Python and SQL, legacy table ACLs

Cluster Access Mode

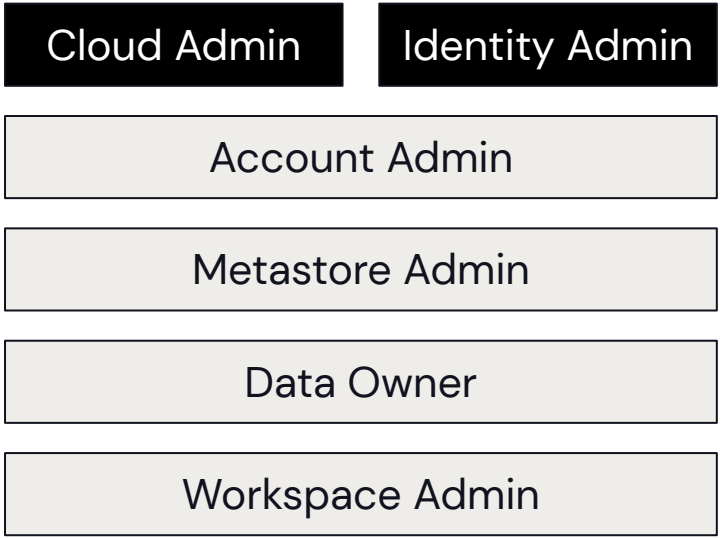
Feature matrix

Access mode	Supported languages	Legacy table ACL	Credential passthrough	Shareable	RDD API	DBFS Fuse mounts	Init scripts and libraries	Dynamic views	Machine learning
No Isolation Shared	All			●	●	●	●		●
Single user	**All**				●	●	●	●	●
Shared	**SQL Python**	●		●				●	

Roles and Identities in Unity Catalog

Unity Catalog

Roles



Cloud Admin

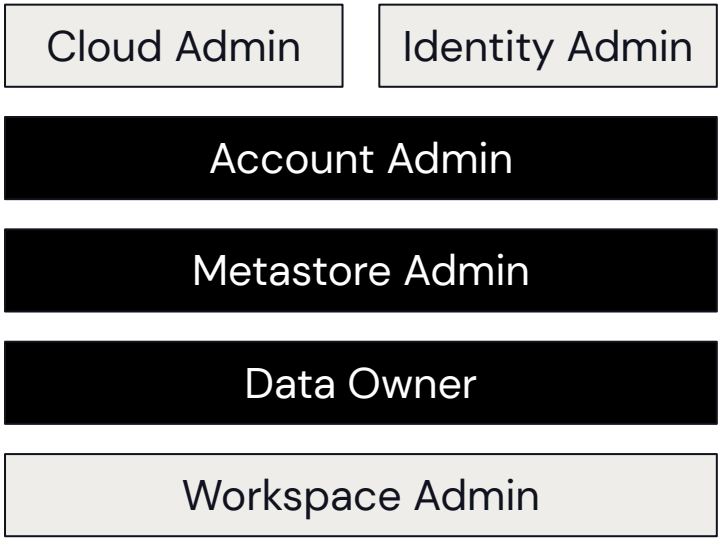
- Manage underlying cloud resources
  - Storage accounts/buckets
  - IAM role/service principals/managed identities

Identity Admin

- Manage users and groups in the identity provider (IdP)
- Provision into account (with account admin)

Unity Catalog

Roles



Account Admin

- Create or delete metastores, assign metastores to workspaces
- Manage users and groups, integrate with IdP
- Full access to all data objects

Metastore Admin

- Create or drop, grant privileges on, and change ownership of catalogs and otherdata objects

Data Owner – owns data objects they created

- Create nested objects, grant privileges on, and change ownership of owned objects

Unity Catalog

Roles

Cloud Admin

Identity Admin

Account Admin

Metastore Admin

Data Owner

Workspace Admin

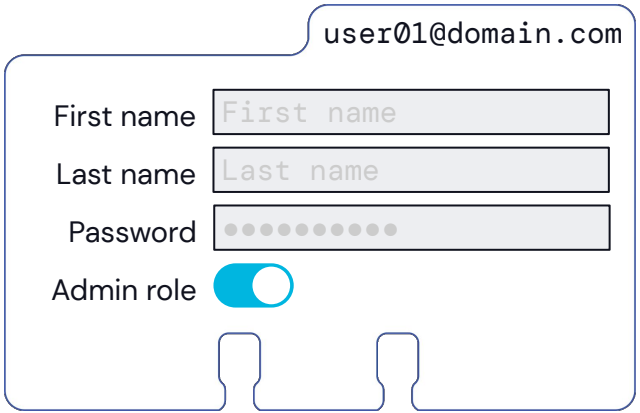
Workspace Admin

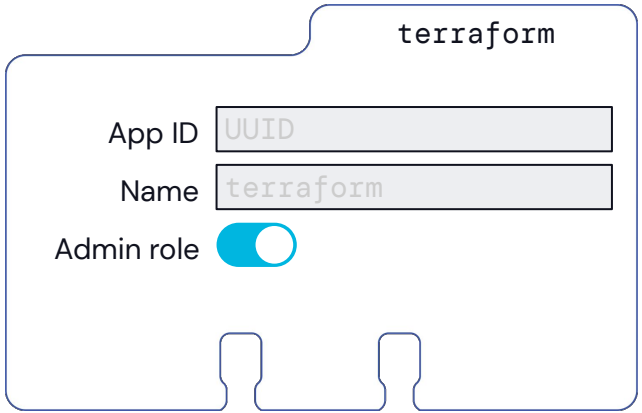
- Manages permissions on workspace assets
- Restricts access to cluster creation
- Adds or removes users
- Elevates users permissions
- Grant privileges to others
- Change job ownership

Unity Catalog

Identities

- User
- Account Administrator
- Service Principal
- Service Principal with administrative privileges

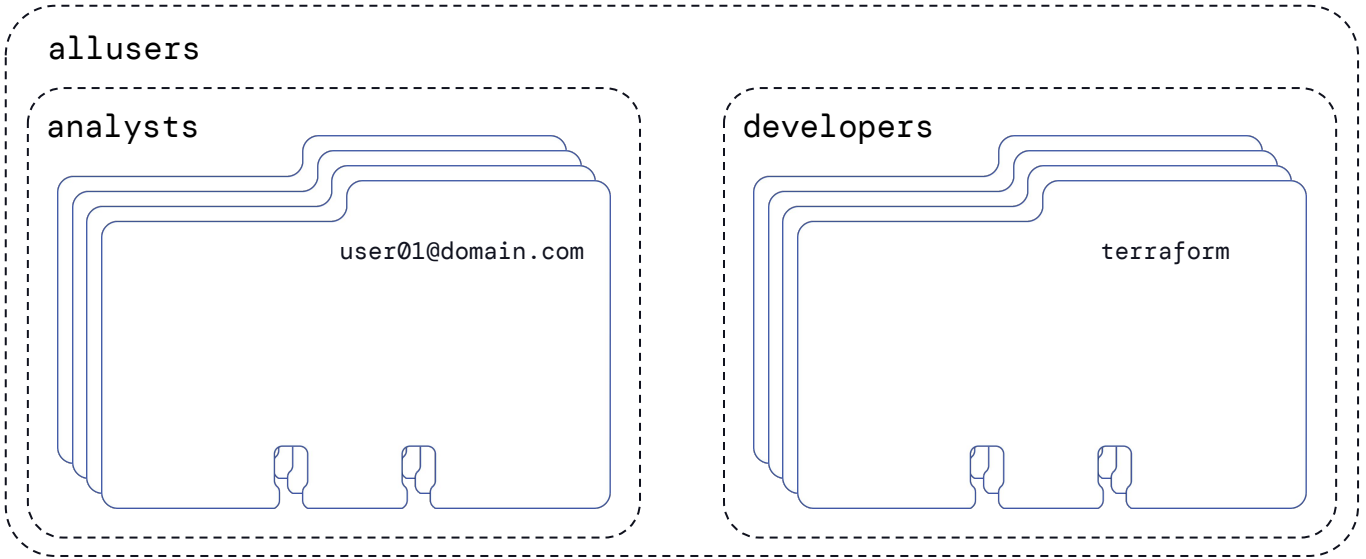




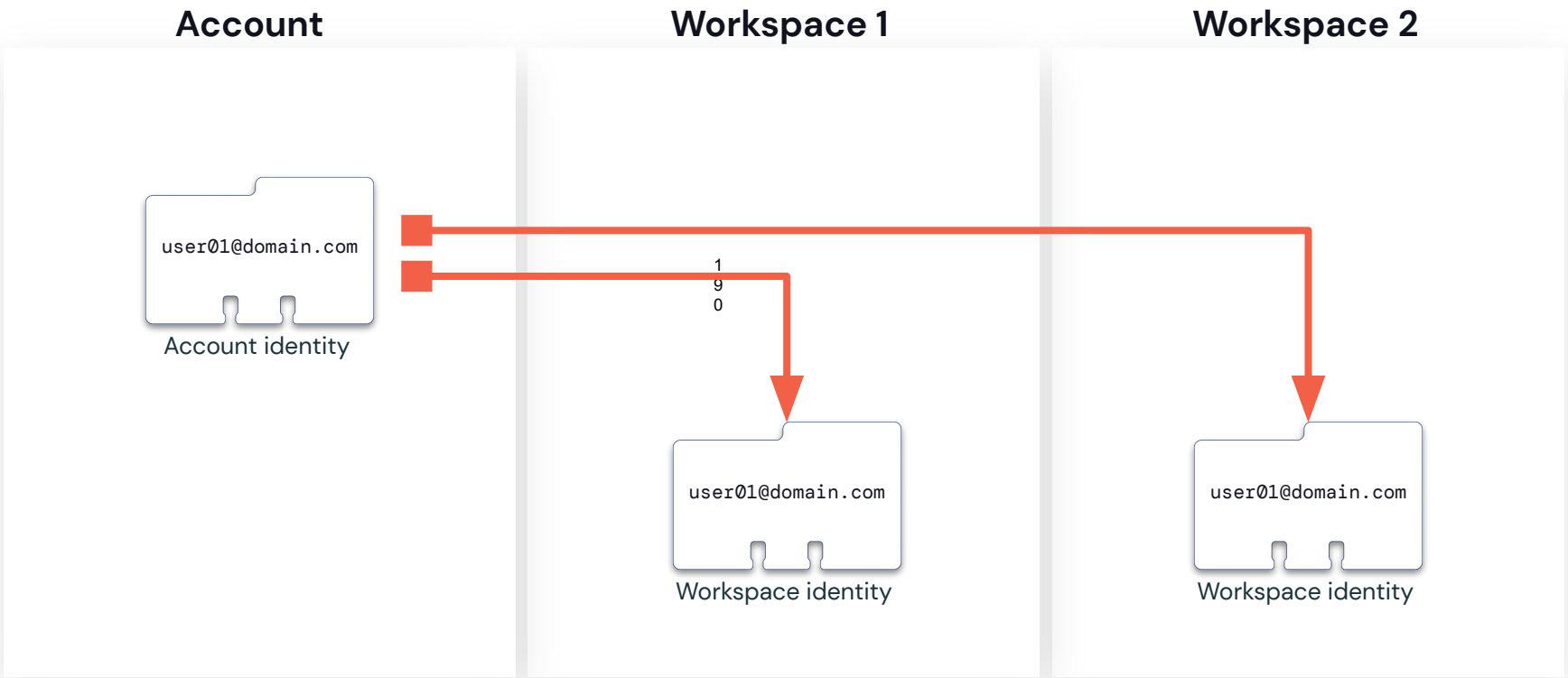
Unity Catalog

Identities

- Groups

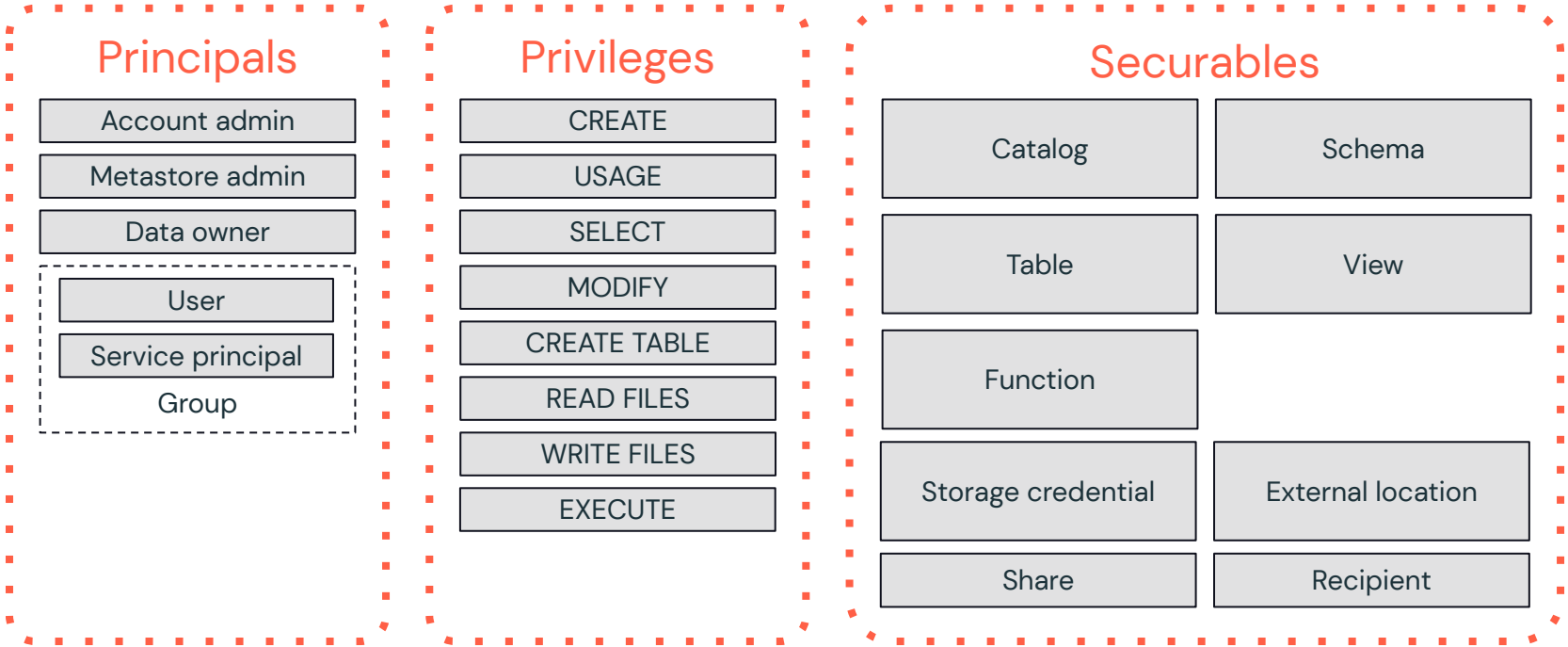


Unity Catalog
Identity Federation

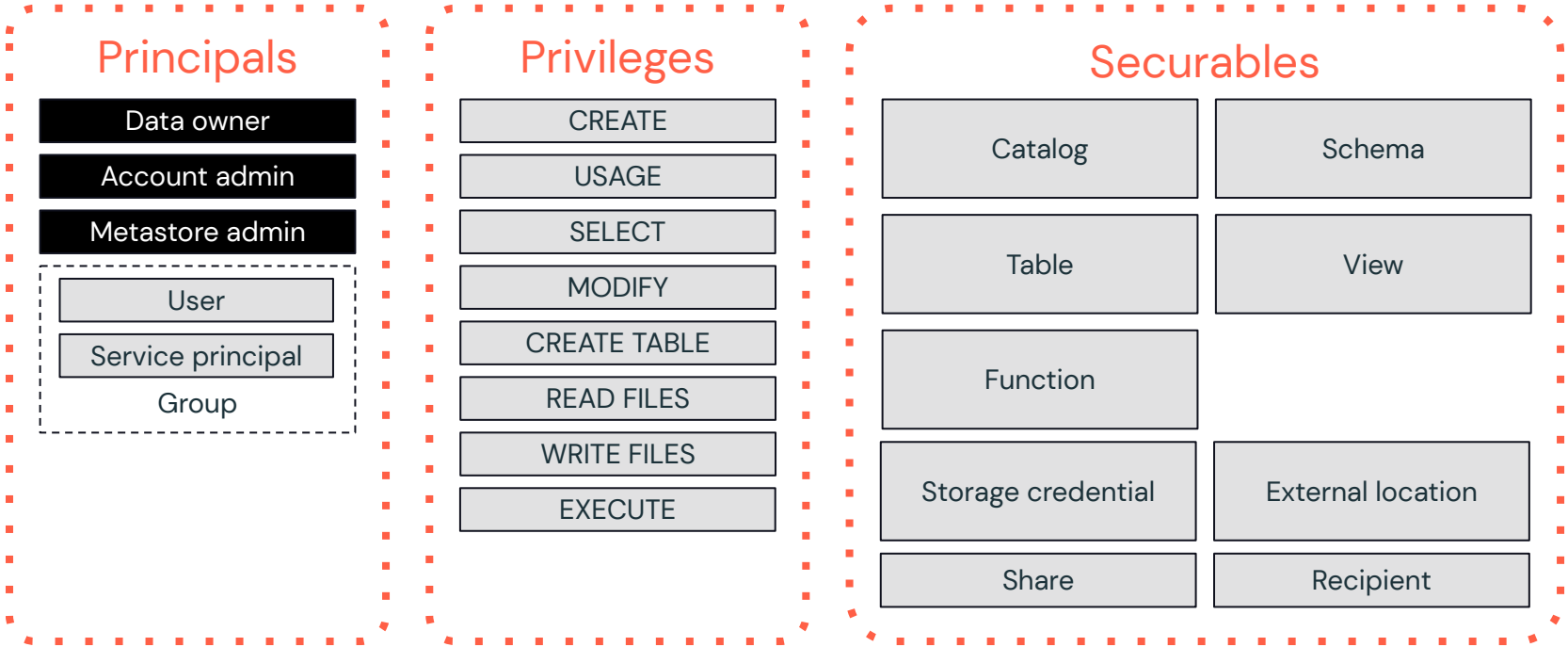


Data Access Control in Unity Catalog

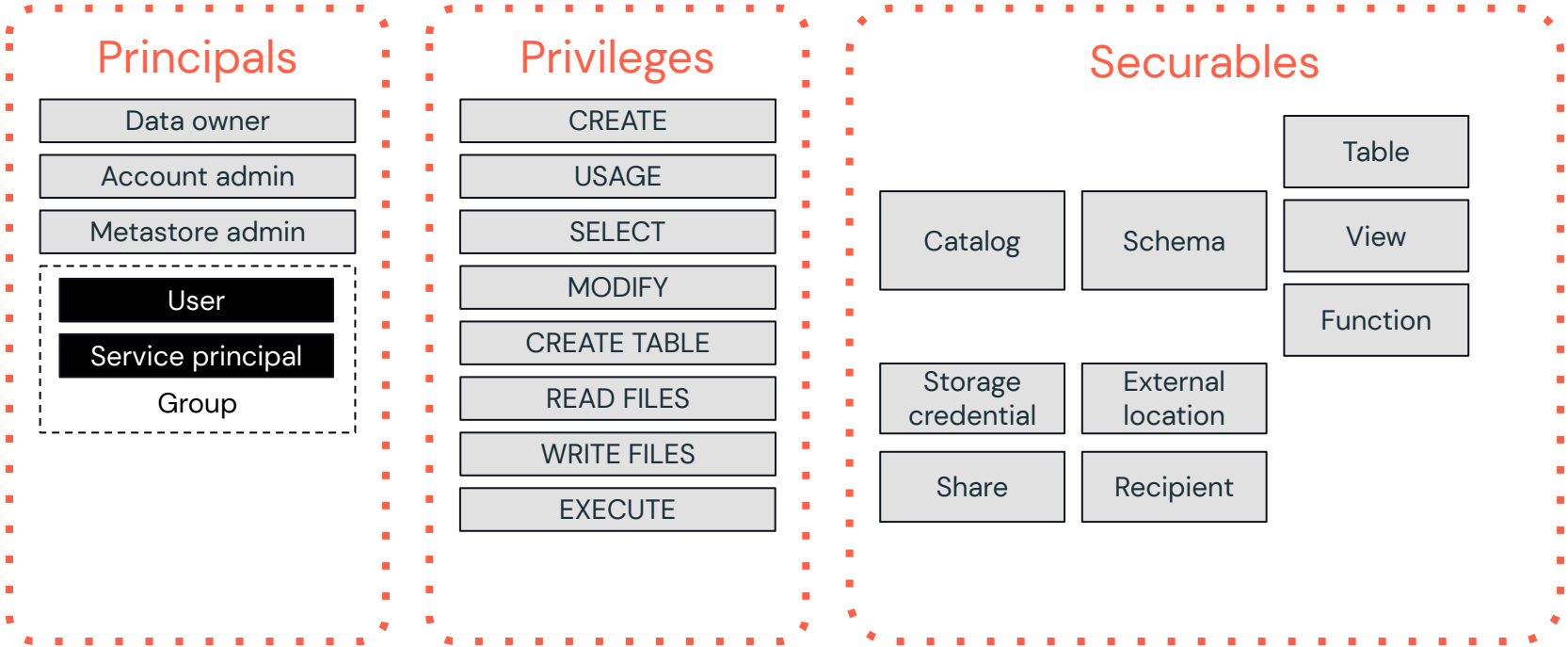
Security model



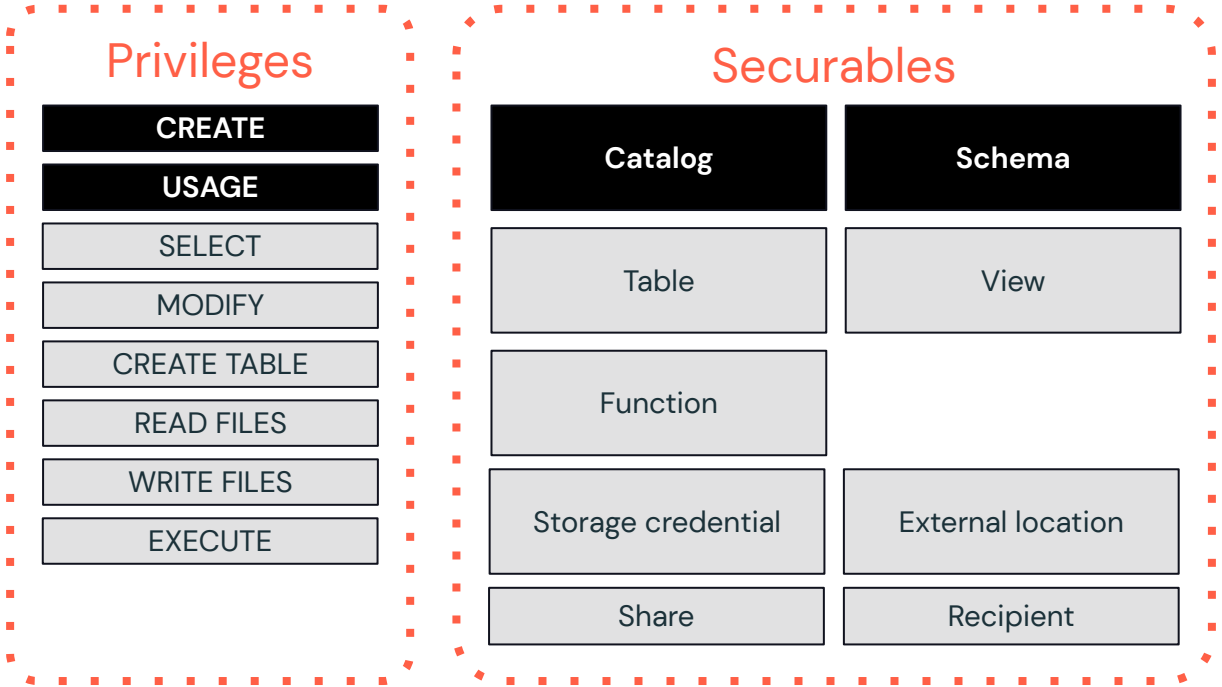
Security model



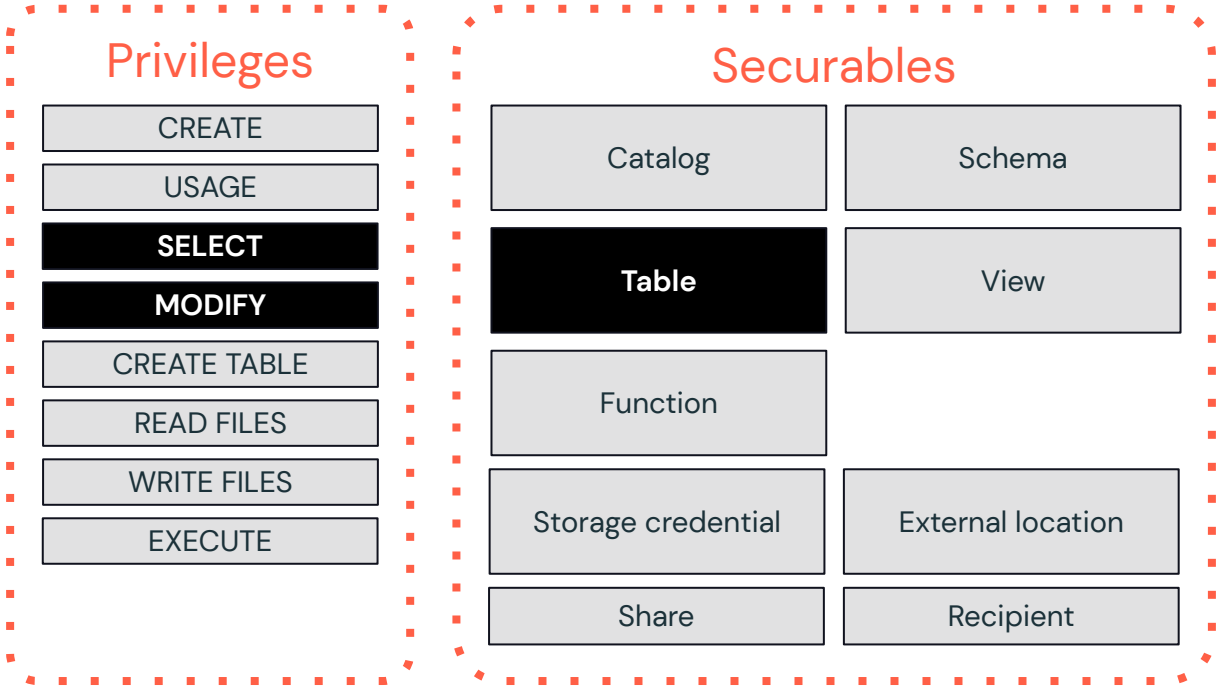
Security model



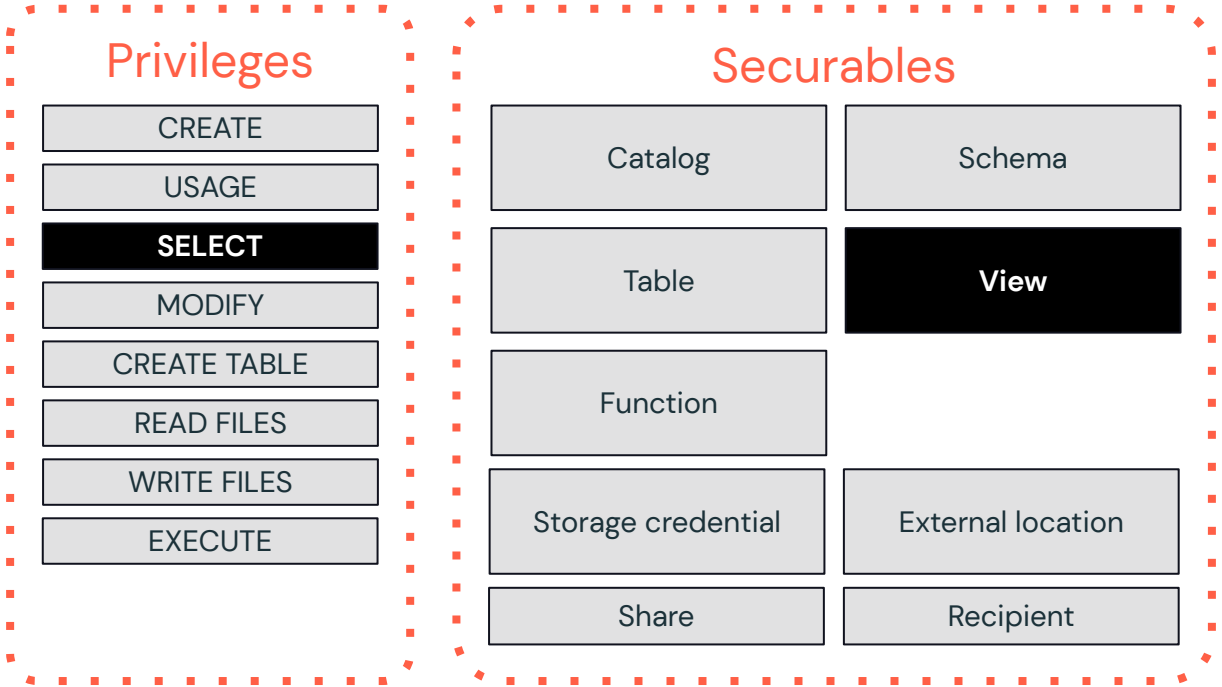
Security model



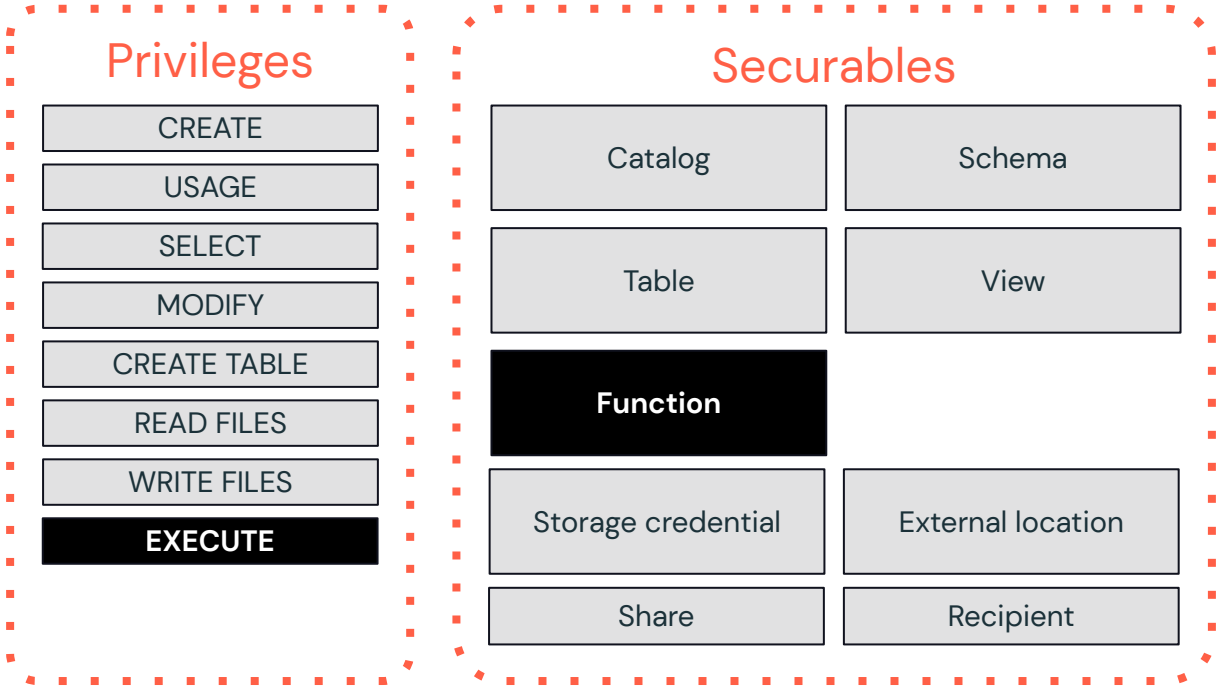
Security model



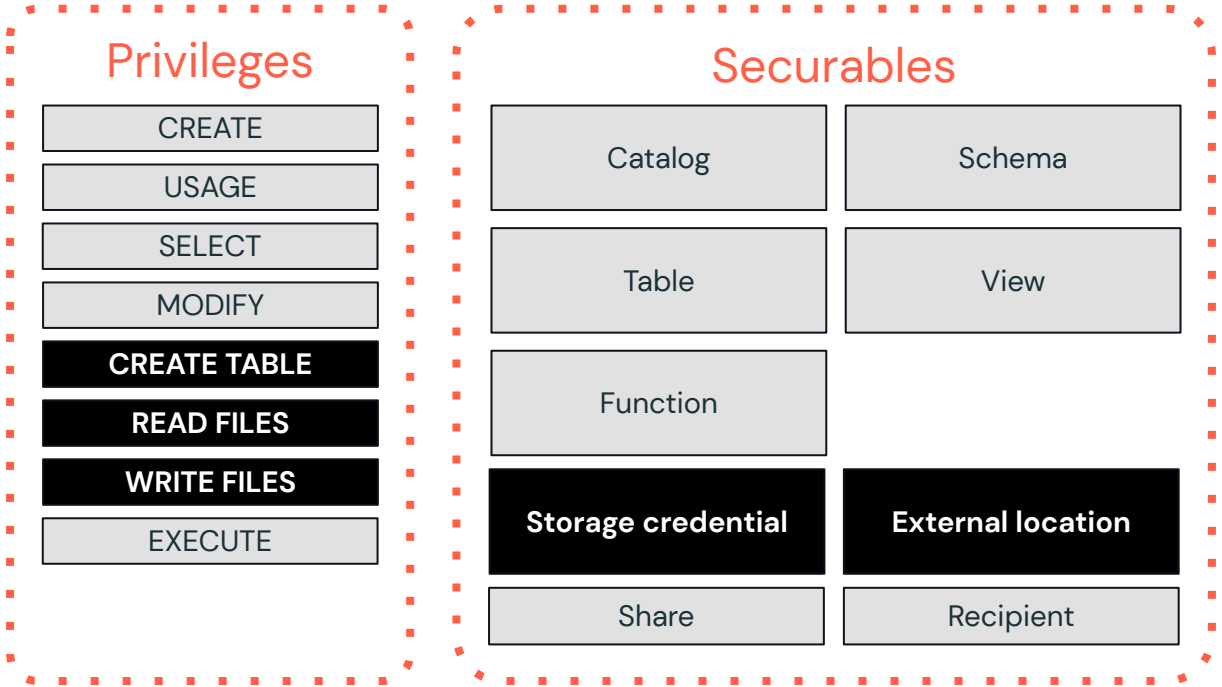
Security model



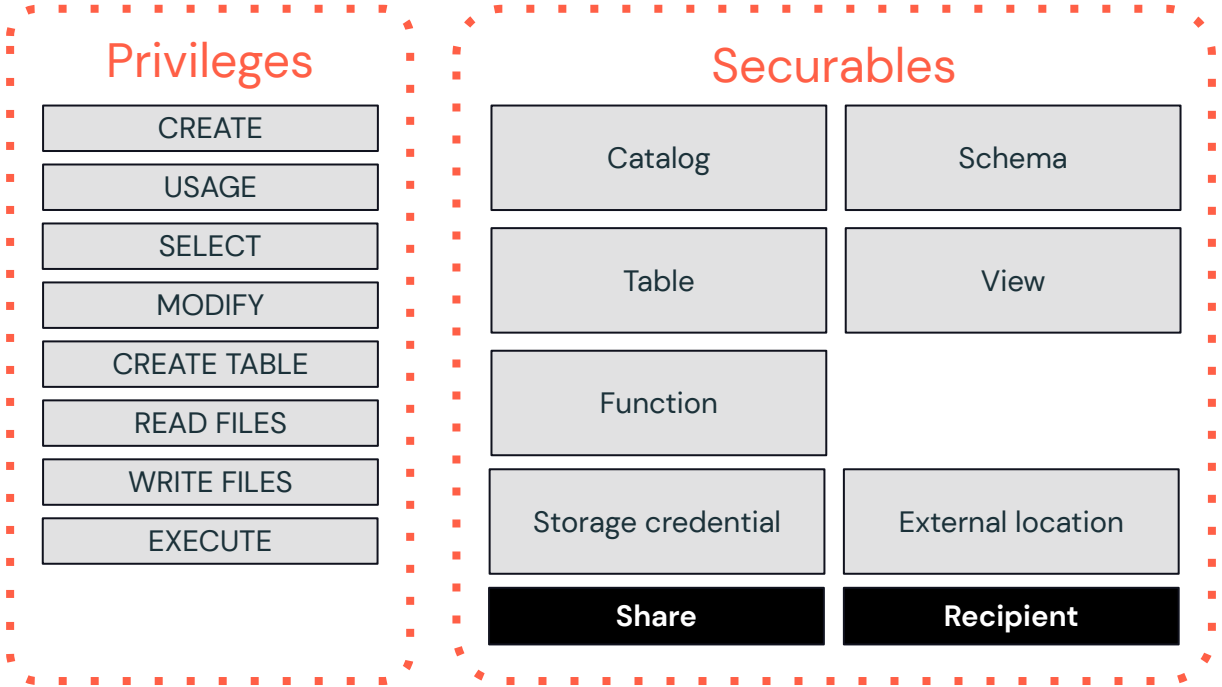
Security model



Security model

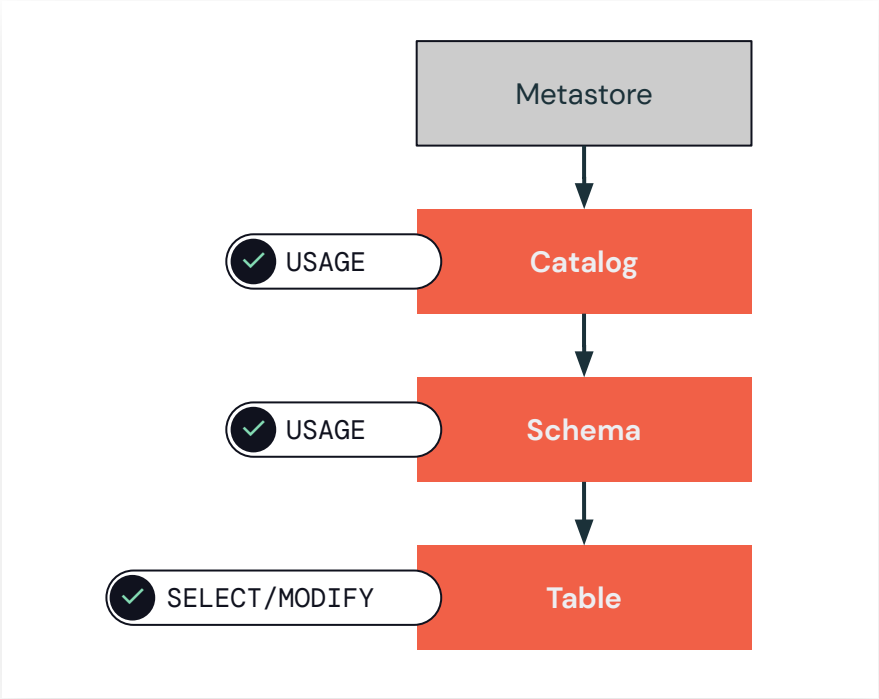


Security model



Privilege Recap

Tables



Querying tables (**SELECT**)

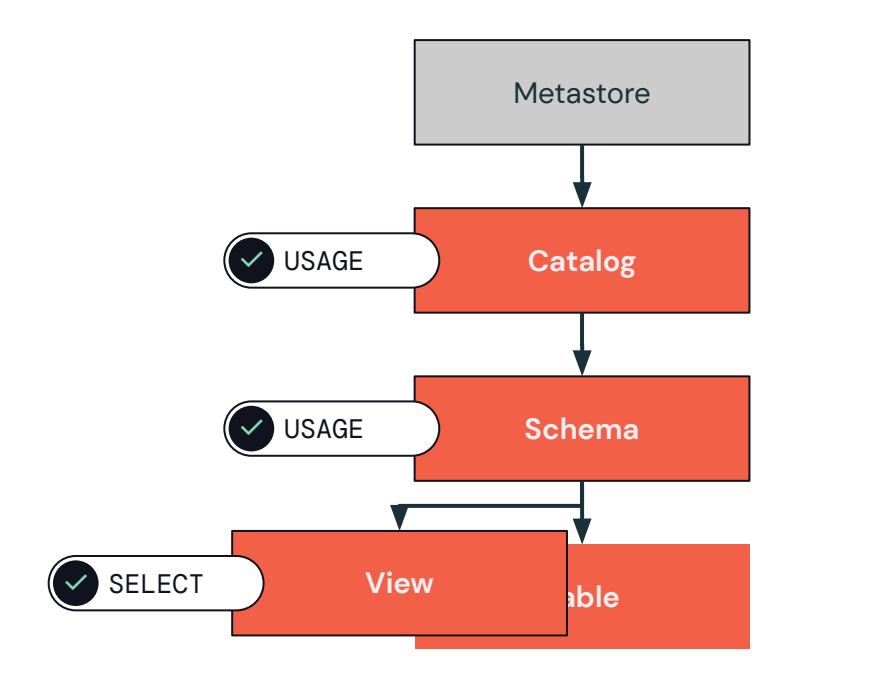
Modifying tables (**MODIFY**)

- Data (INSERT, DELETE)
- Metadata (ALTER)

Traversing containers (**USAGE**)

Privilege Recap

Views



Abstract complex queries

- Aggregations
- Transformations
- Joins
- Filters

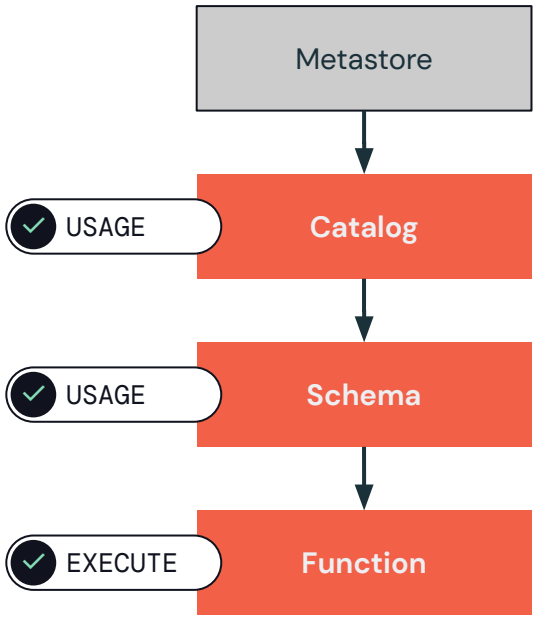
Enhanced table access control

Querying views (**SELECT**)

Traversing containers (**USAGE**)

Privilege Recap

Functions



Provide custom code via user-defined functions

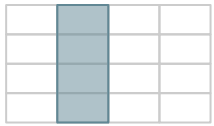
Using functions (**EXECUTE**)

Traversing containers (**USAGE**)

Dynamic Views

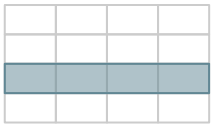
Limit access to columns

Omit column values from output



Limit access to rows

Omit rows from output



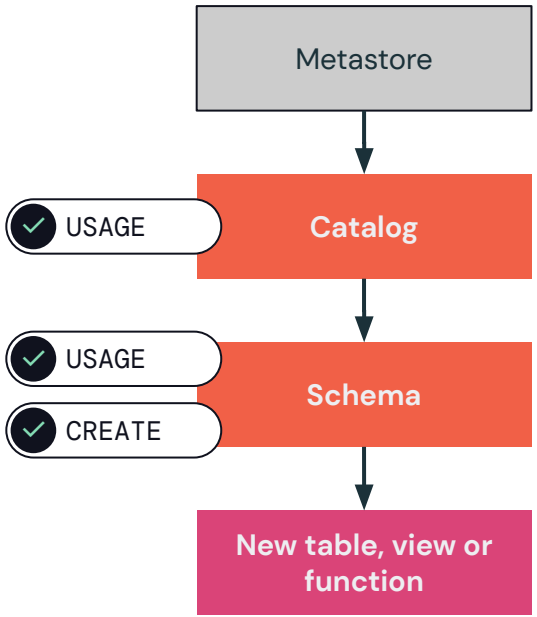
Data Masking

Obscure data

●●●●●●@databricks.com

Can be conditional on a specific user/service principal or group membership through Databricks-provided functions

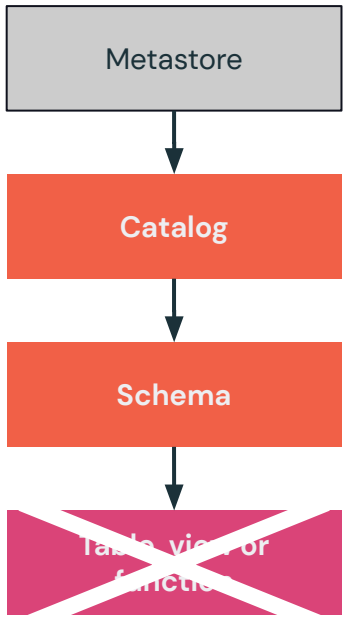
Creating New Objects



Creating new objects (**CREATE**)

Traversing containers (**USAGE**)

Deleting Objects



DROP objects

Unity Catalog External Storage

Storage Credentials and External Locations

Storage Credential

Enables Unity Catalog to connect to an external cloud storage

Examples include:

- IAM role for AWS S3
- Service principal for Azure Storage

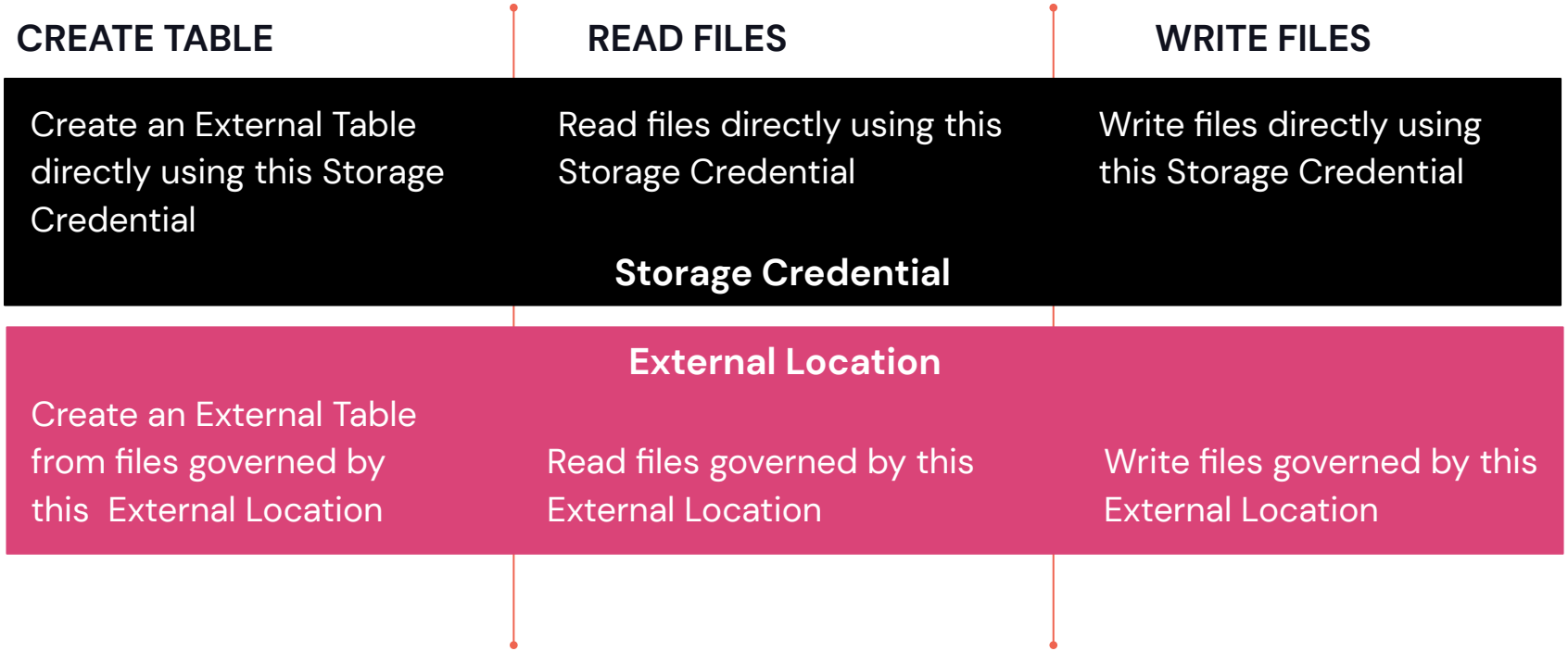
External Location

Cloud storage path + storage credential

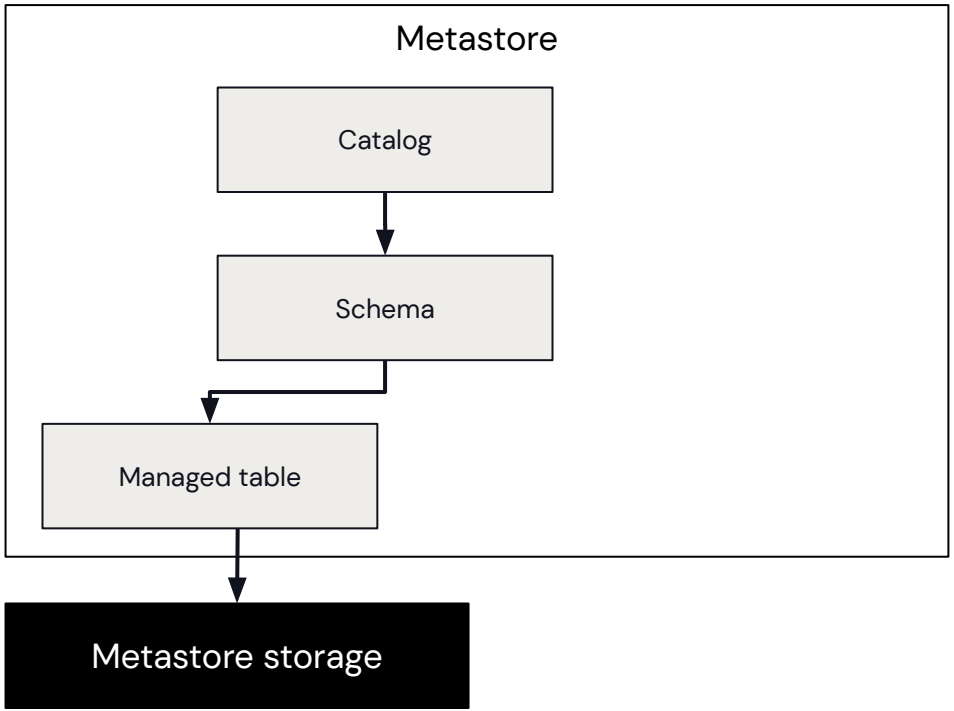
- Self-contained object for accessing specific locations in cloud storage
- Fine-grained control over external storage

Storage Credentials and External Locations

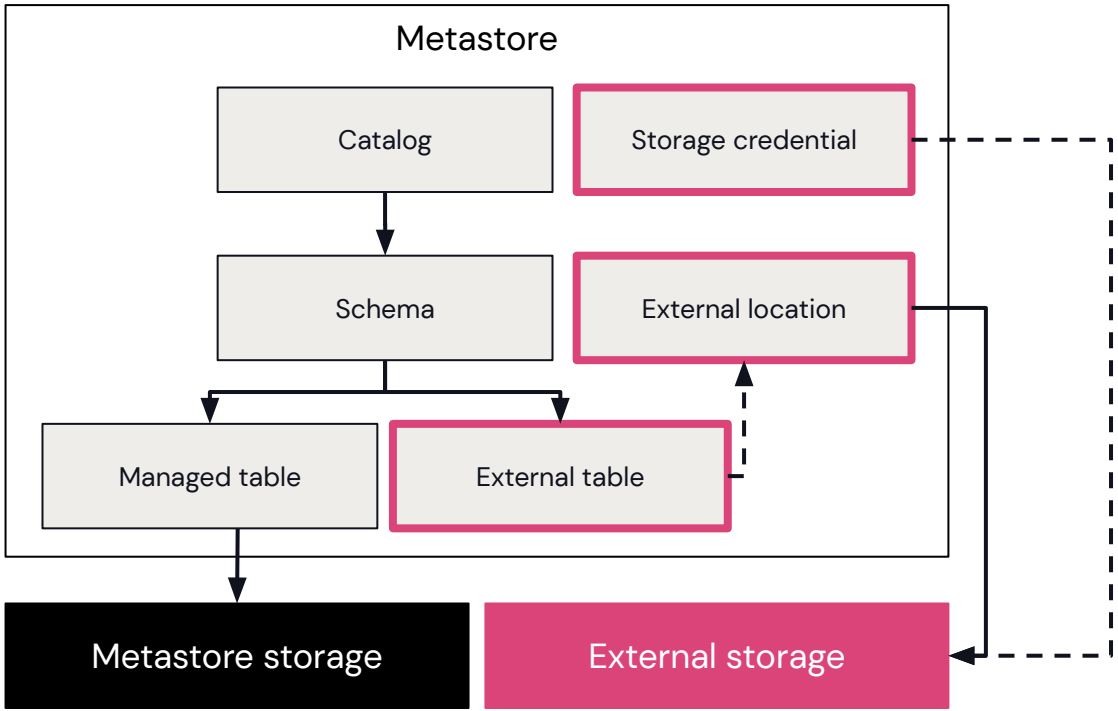
Access Control



Managed Tables



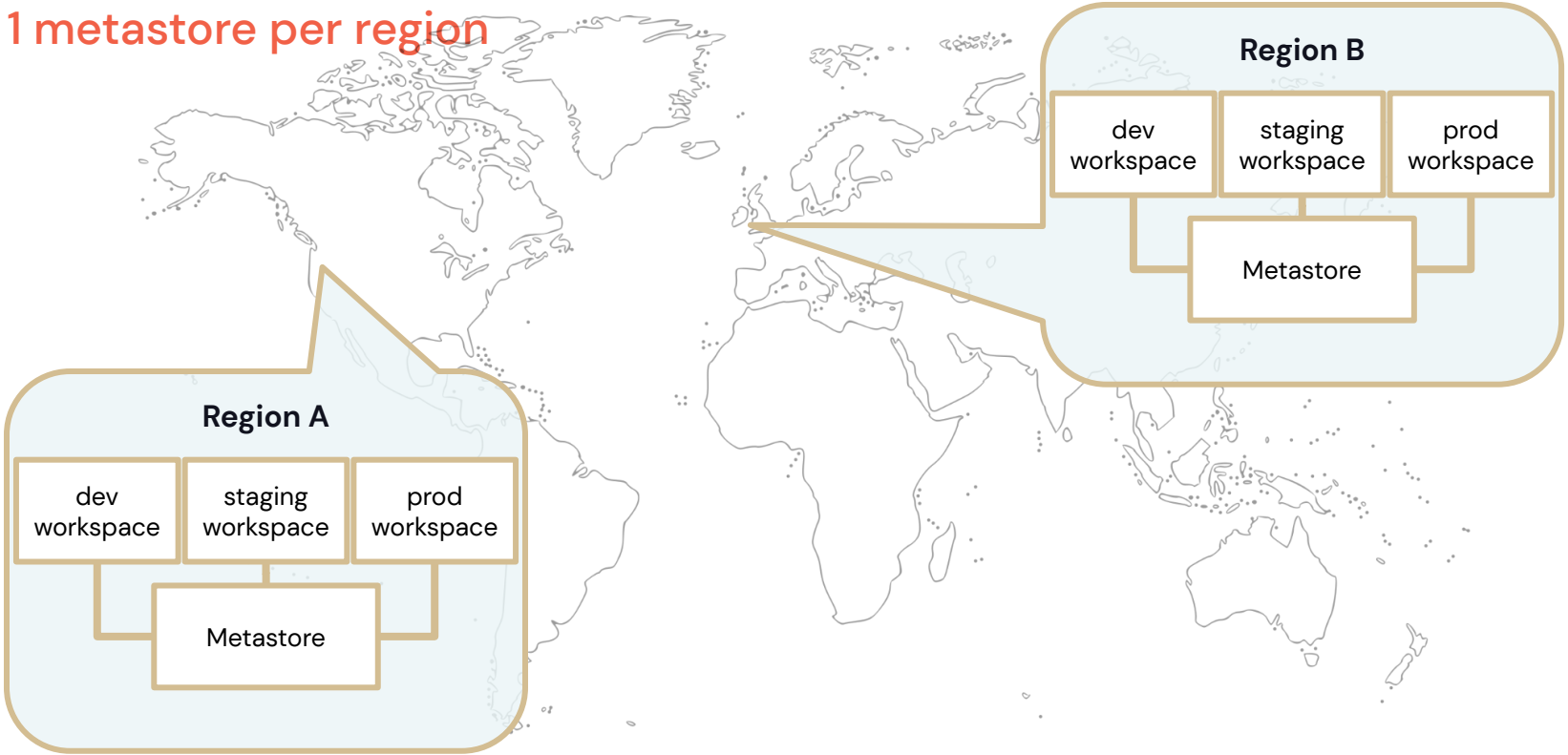
External Tables



Unity Catalog Patterns and Best Practices

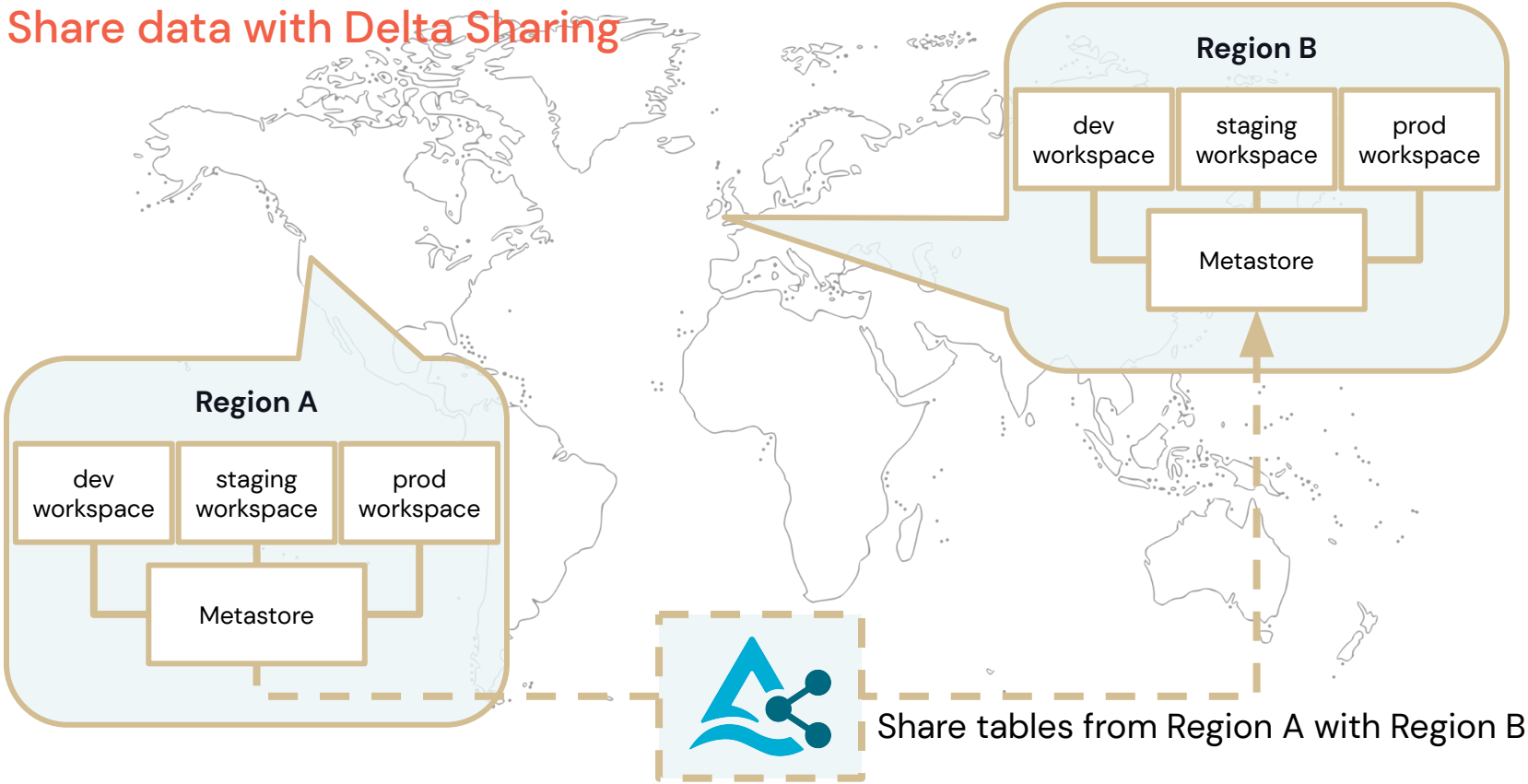
UC Patterns & Best Practices

1 metastore per region



UC Patterns & Best Practices

Share data with Delta Sharing



UC Patterns & Best Practices

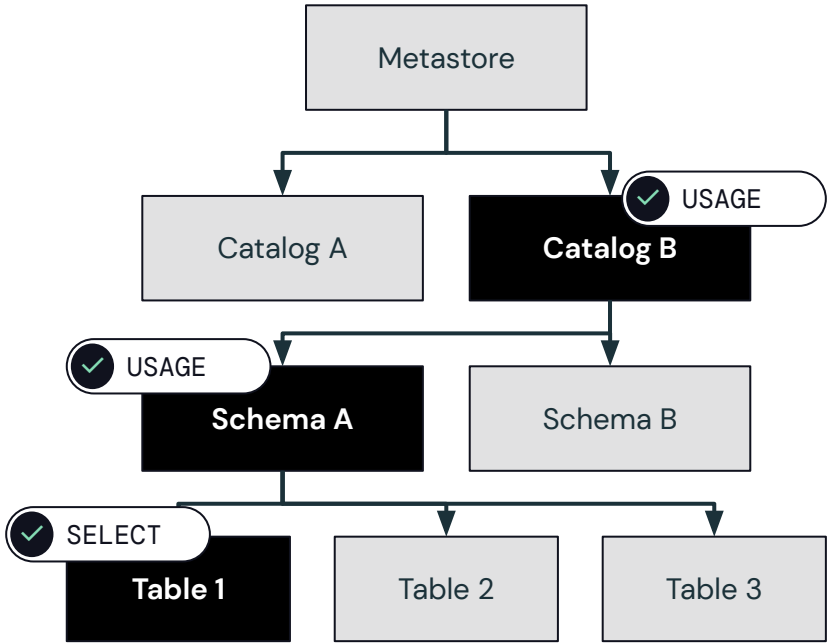
Data Segregation

Use **catalogs** (not metastores) to segregate data

Apply permissions appropriately

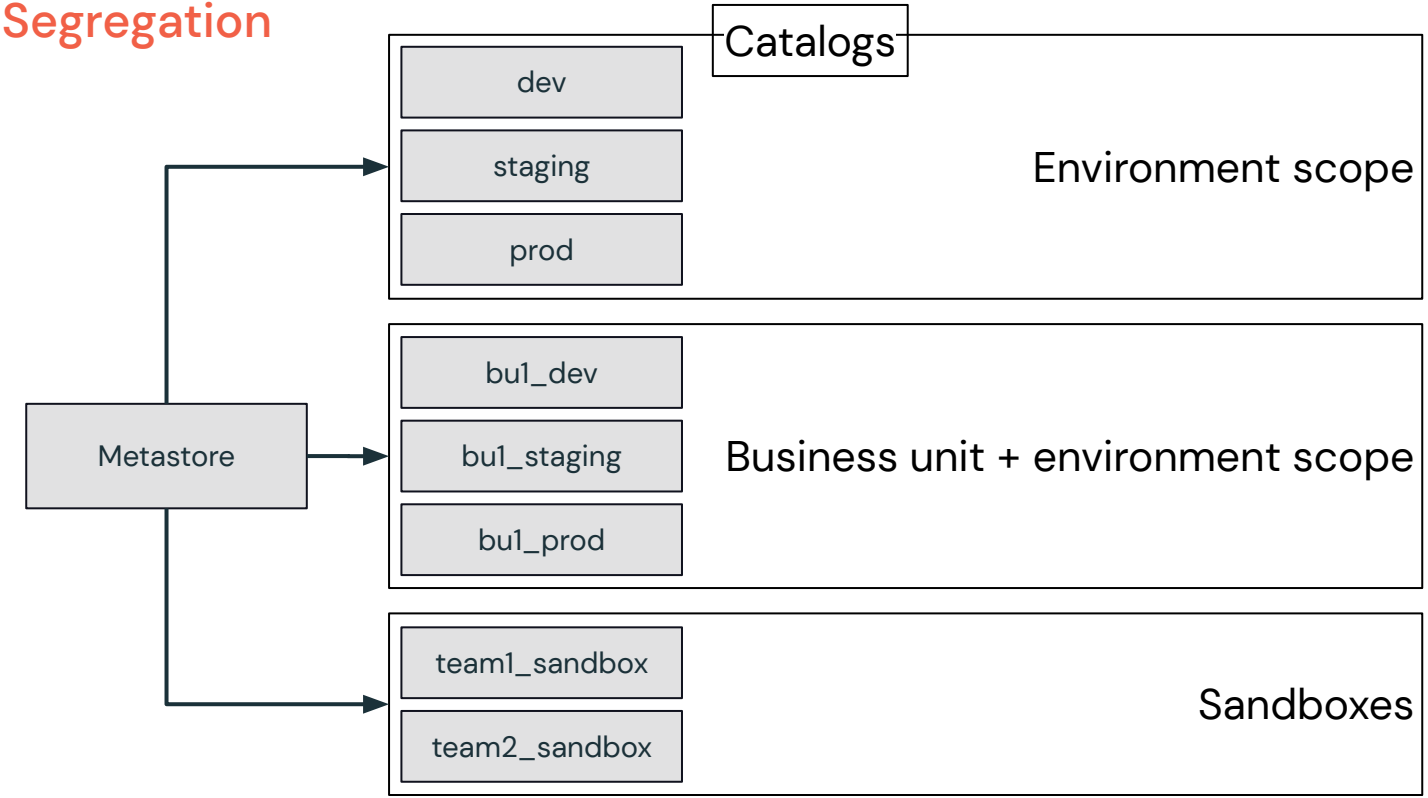
For example, grant to group B:

- **USAGE** on catalog B
- **USAGE** on all applicable schemas in catalog B
- **SELECT/MODIFY** on applicable tables



UC Patterns & Best Practices

Data Segregation



UC Patterns & Best Practices

Identity Management

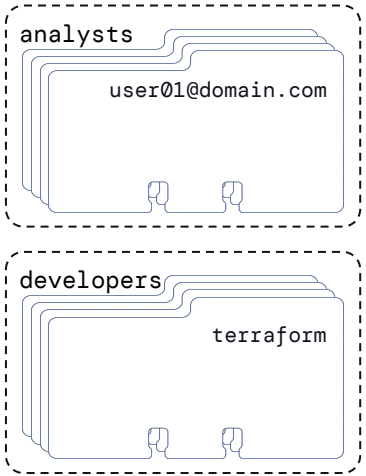
Account-level Identities

Manage all identities at the account-level

Enable UC for workspaces to enable identity federation

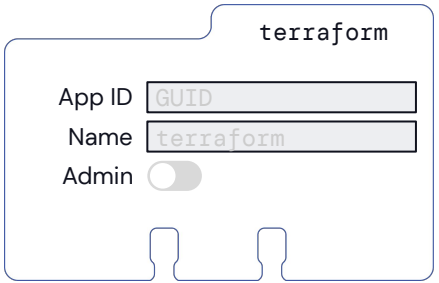
Groups

Use groups rather than users to assign access and ownership to securable objects



Service Principals

Use service principals to run production jobs



UC Patterns & Best Practices

Storage Credentials and External locations

Storage Credential

Enables Unity Catalog to connect to an external cloud store

Examples include:

- IAM role for AWS S3
- Service principal for Azure Storage

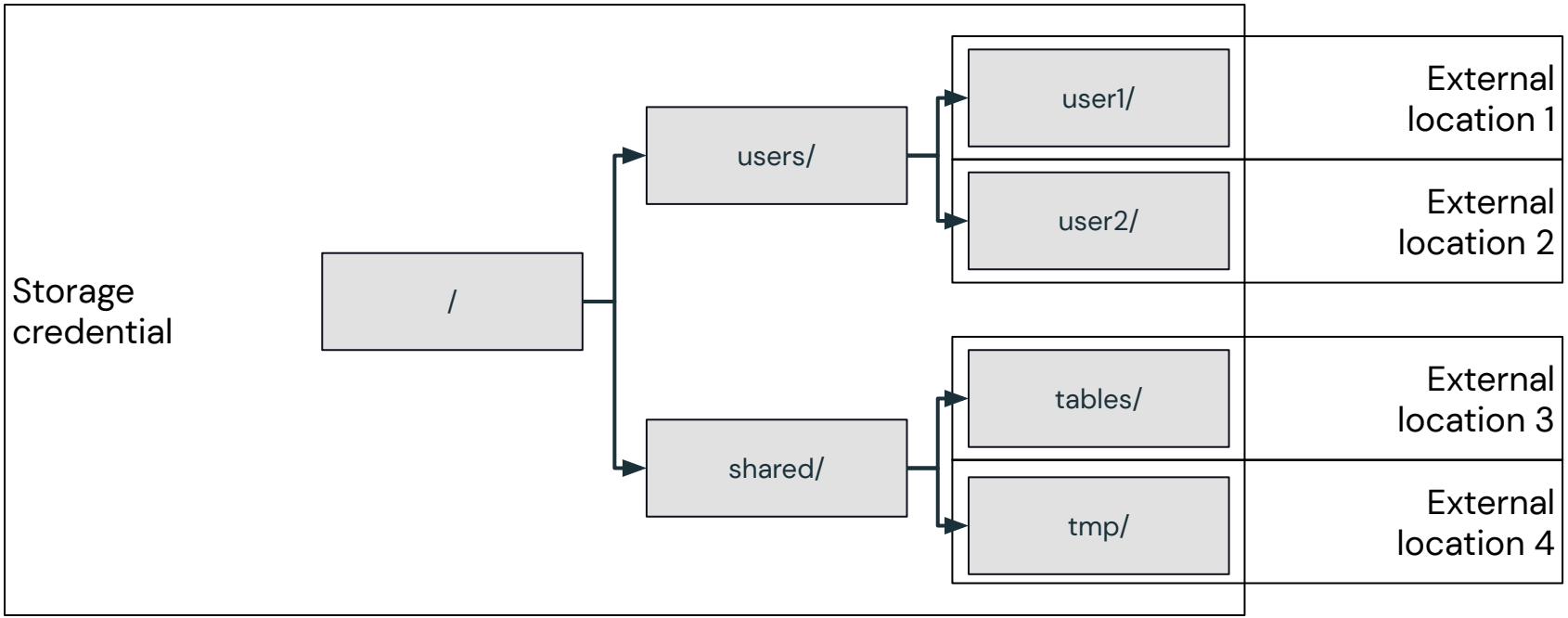
External Location

Cloud storage path + storage credential

- Self-contained object for accessing specific locations in cloud stores
- Fine-grained control over external storage

UC Patterns & Best Practices

Storage Credentials and External Locations



UC Patterns & Best Practices

Managed versus External Tables

Managed Tables

Metadata lives in control plane

Data lives in metastore-managed storage location

DROP discards data

Delta format only

External Tables

Metadata lives in control plane

Data lives in user-provided storage location (external to UC)

DROP leaves data intact

Several formats supported (delta, csv, json, avro, parquet, orc, text)

UC Patterns & Best Practices

When to use external tables?

Quick and easy upgrade from external table in Hive metastore

External readers or writers

Requirement for specific storage naming or hierarchy

Infrastructure-level isolation requirements

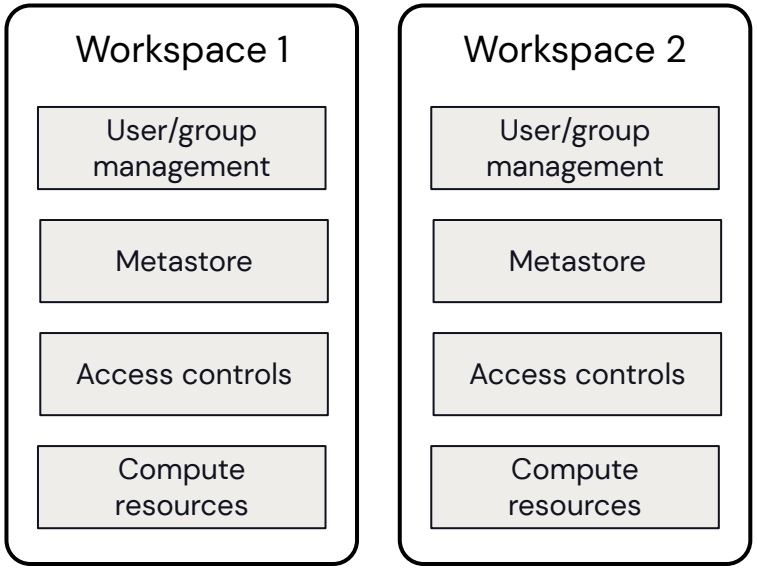
Non-Delta support requirement

Unity Catalog Key Capabilities

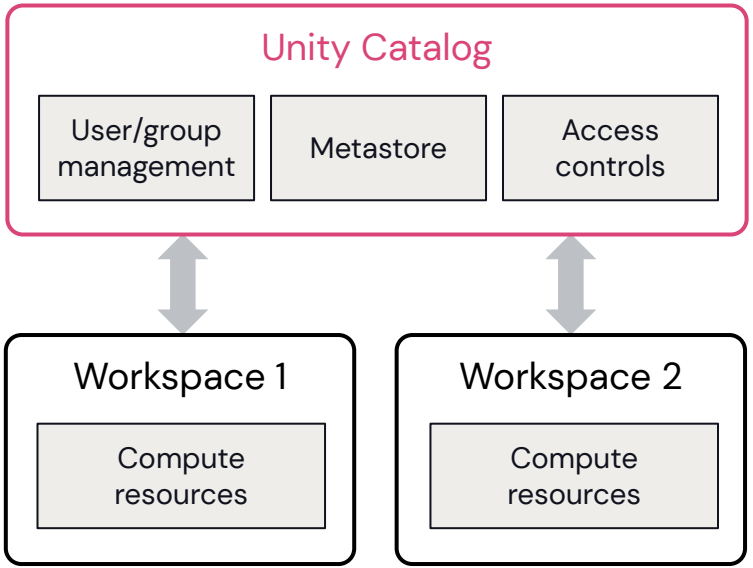
Centralized metadata and user management

Unity Catalog Architecture

Before Unity Catalog



With Unity Catalog



Centralized Access Controls

Centrally grant and manage access permissions across workloads

Using ANSI SQL DCL

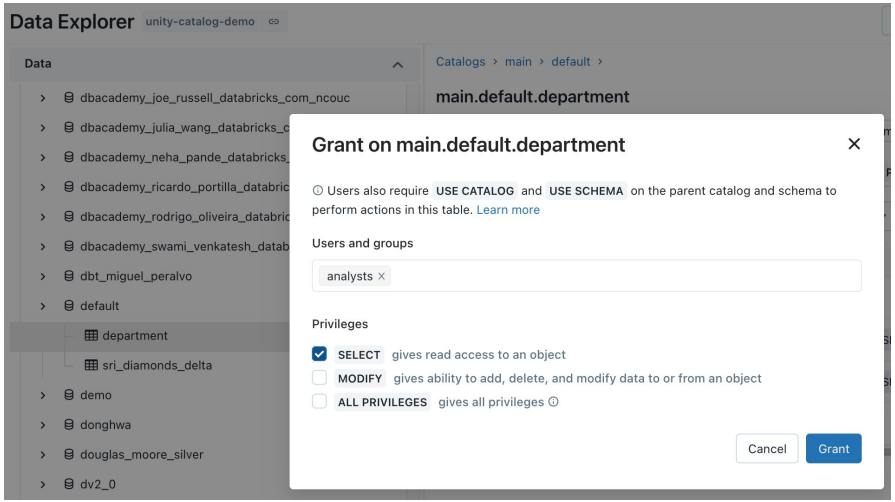
```
GRANT <privilege> ON <securable_type>
<securable_name> TO `<principal>`

GRANT SELECT ON iot.events TO engineers
```

Choose permission level

'Table'= collection of files in S3/ADLS

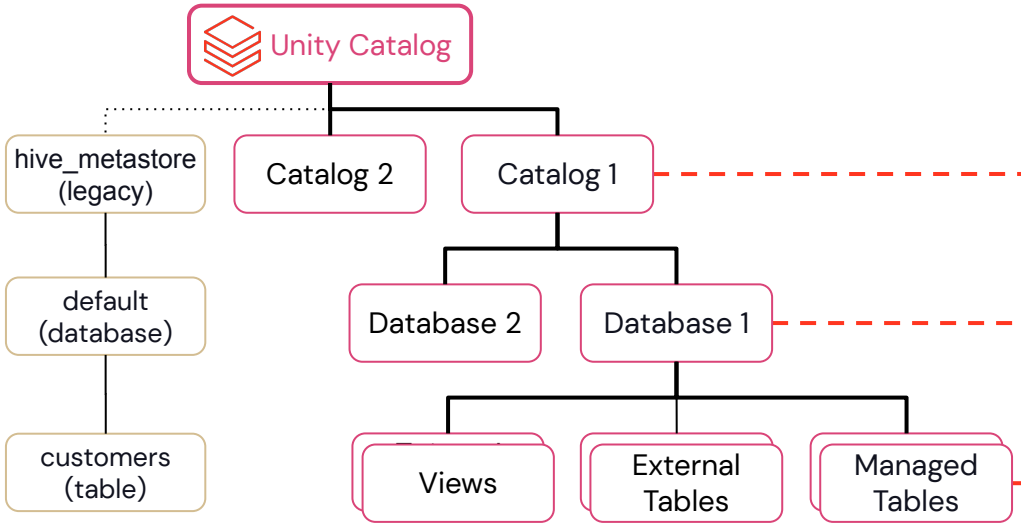
Sync groups from your identity provider

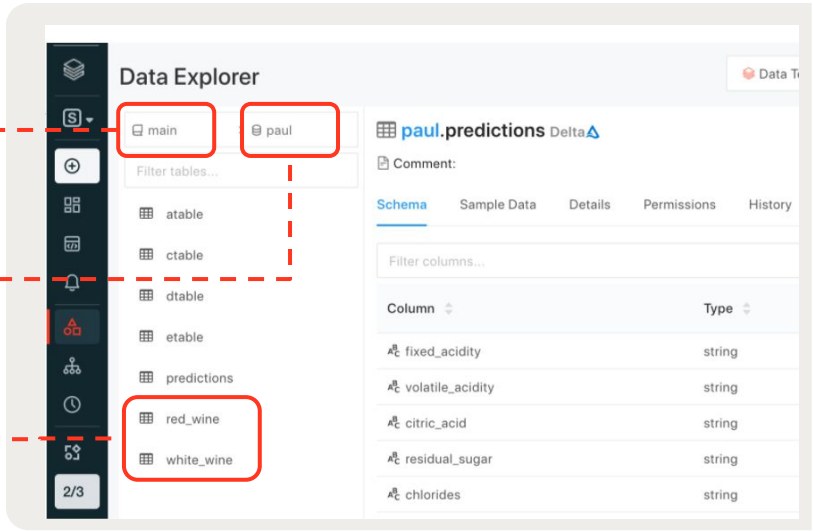
Using UI



Three level namespace

Seamless access to your existing metastores

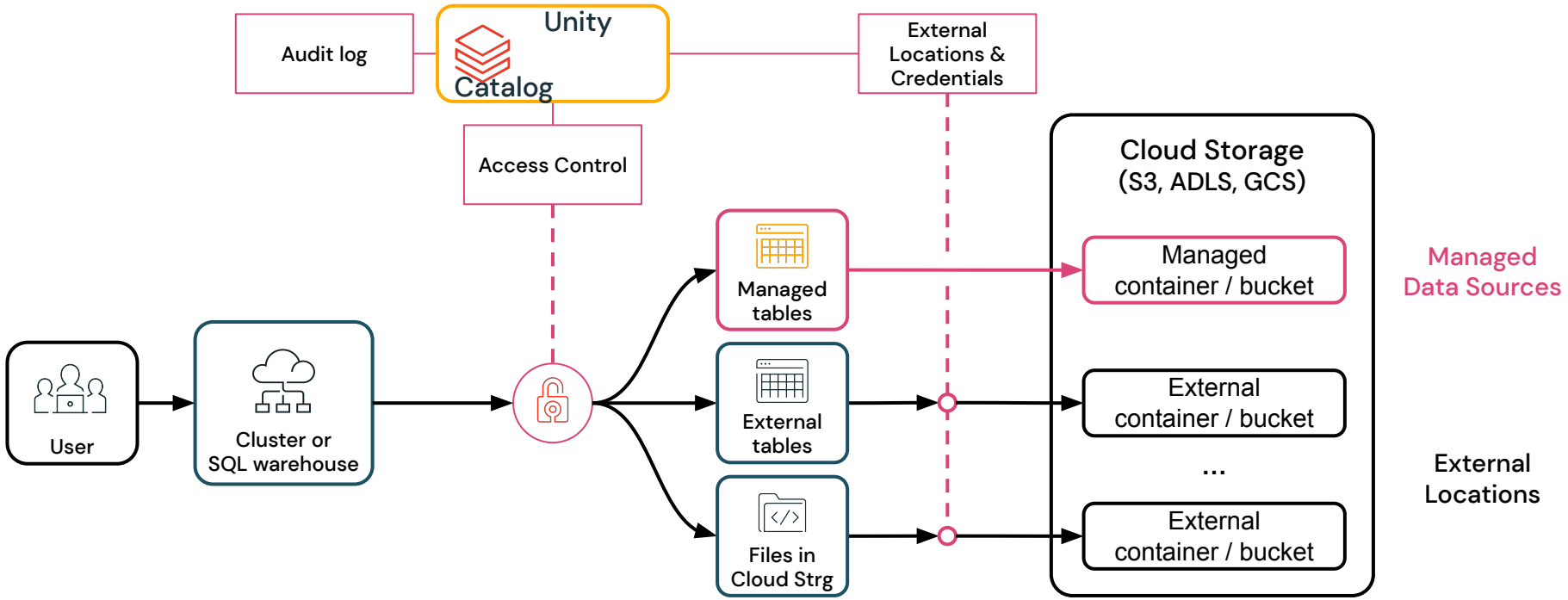





```
SELECT * FROM main.student.example;  -- <catalog>.<database>.<table>

SELECT * FROM hive_metastore.default.customers;
```

Managed Data Sources & External Locations

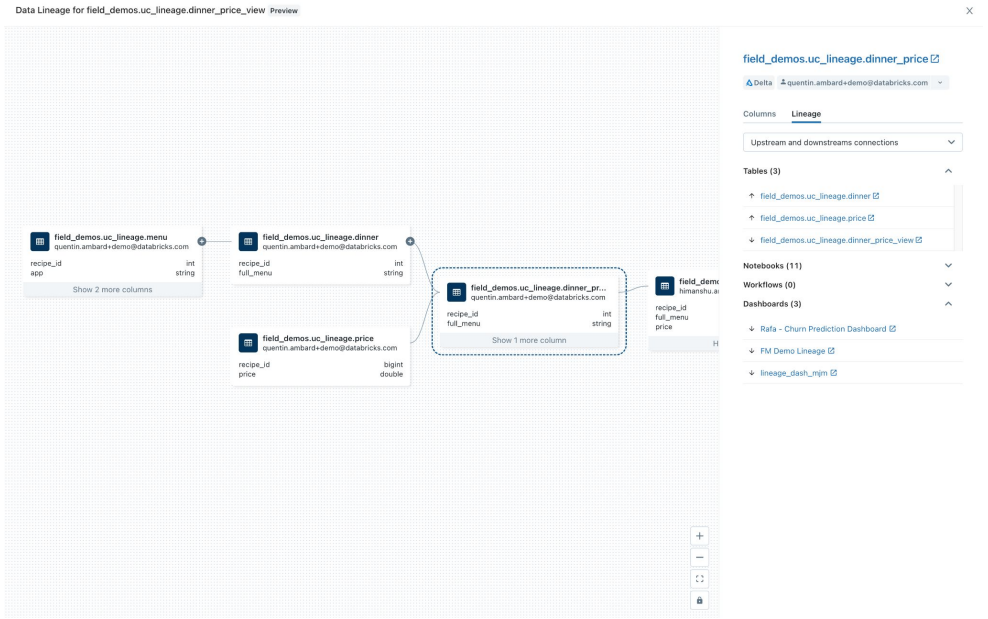
Simplify data access management across clouds



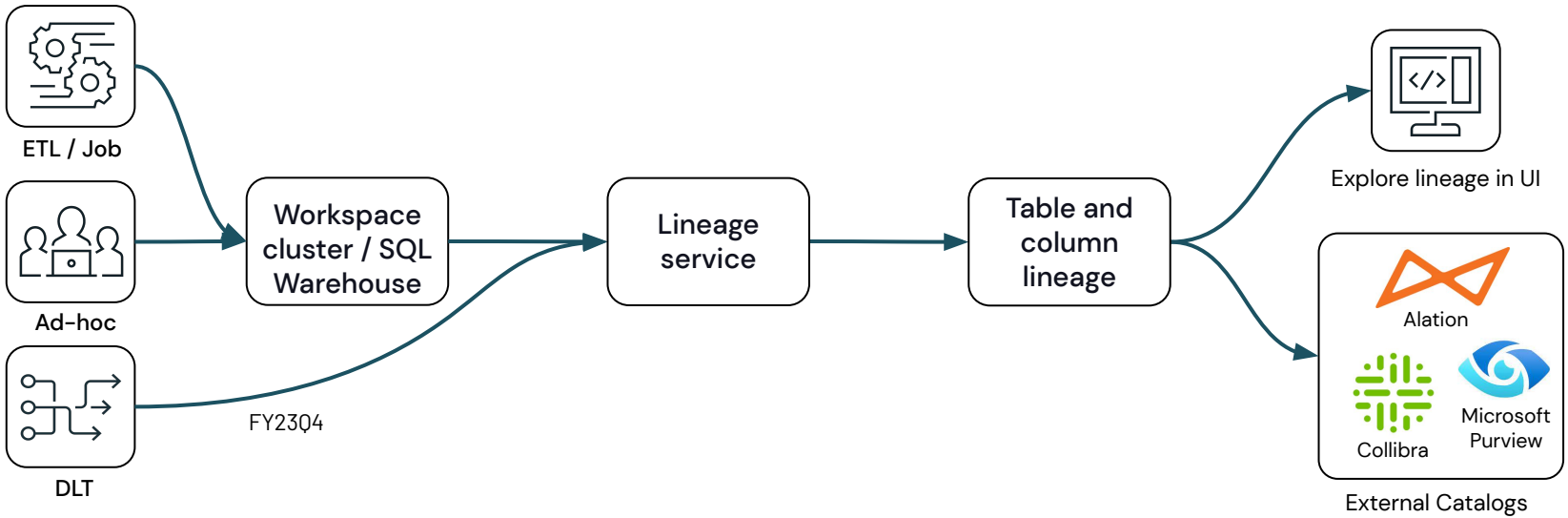
Automated lineage for all workloads

End-to-end visibility into how data flows and consumed in your organization

- Auto-capture runtime data lineage on a Databricks cluster or SQL warehouse
- Track lineage down to the table and column level
- Leverage common permission model from Unity Catalog
- Lineage across tables, dashboards, workflows, notebooks



Lineage flow – How it works



- Code (any language) is submitted to a cluster or SQL warehouse or DLT* executes data flow

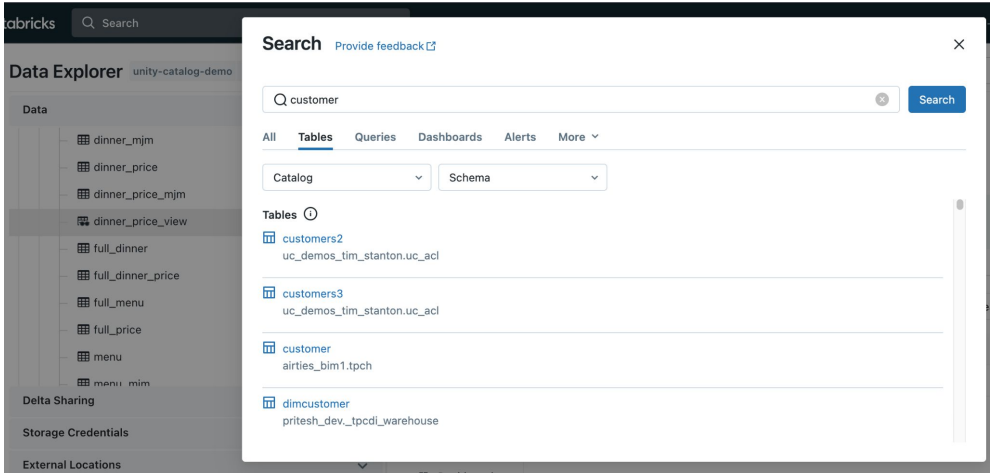
- Lineage service analyzes logs emitted from the cluster, and pulls metadata from DLT
- Assembles column and table level lineage

- Presented to the end user graphically in Databricks
- Lineage can be exported via API and imported into other tool

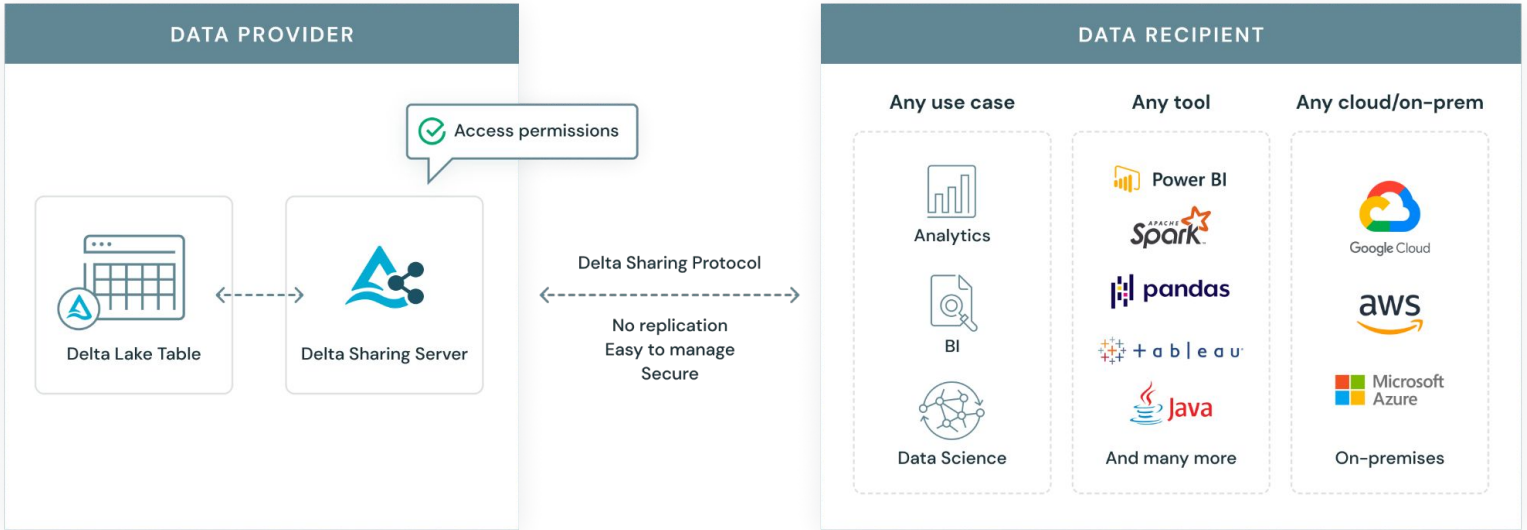
Built-in search and discovery

Accelerate time to value with low latency data discovery

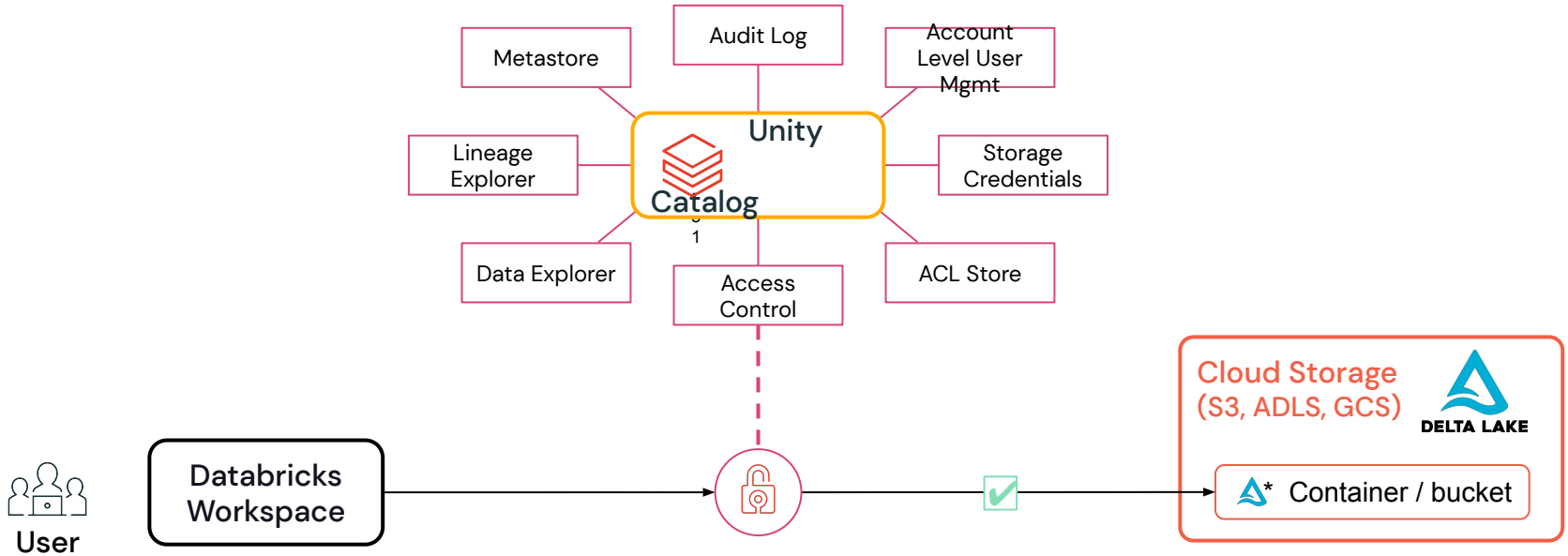
- UI to search for data assets stored in Unity Catalog
- Unified UI across DSML + DBSQL
- Leverage common permission model from Unity Catalog



An open standard for secure sharing of data assets



Unity Catalog -Architecture



* Unity Catalog will support any data format (table or raw files)


databricks