Databricks Management

Databricks Workspace

A **Workspace in Databricks** is the **collaboration, development, and execution environment** where users write code, run jobs, work with notebooks, build pipelines, and access compute resources.

It is the **central place** where engineers, analysts, and data scientists interact with Databricks on a daily basis.

It provides:

- Notebooks (Python, SQL, Scala)
- Repos (Git integration)
- Clusters and SQL warehouses (compute)
- Jobs & Workflows (ETL pipelines)
- MLflow tracking
- Workspace folders, dashboards
- Role-based access for compute & workspace objects

Workspace controls:

- Compute access
- Notebook permissions
- Job execution
- Library installation

Workspace does NOT control:

- Table permissions
- Data access governance
- Storage credentials
- These belong to Unity Catalog.

Databricks Unity Catalog

Unity Catalog in Databricks is the **central, unified data governance and security layer** for managing **all data asset -** tables, files, views, volumes, credentials, and machine learning models—across **multiple Databricks workspaces**.

It provides **fine-grained access control**, **data lineage**, **auditing**, and **central metadata management**

It provides:

- Centralized data access control
- Catalog → schema → table organization
- Permissions for SELECT/INSERT/UPDATE/DELETE
- Row/column-level security
- Data lineage
- Metadata management
- External storage governance

UC controls:

- Catalogs (databases)
- Schemas
- Tables & views
- External locations (ADLS/S3/GCS)
- Storage credentials
- Row/column masking
- Auditing & lineage

UC does NOT handle:

- Compute
- Notebook execution
- Workspace operations

Databricks Unity Catalog & Workspace Rules

- Workspace can attach to only **one UC metastore**
- One UC metastore can attach to **multiple workspaces**
- Multiple UC metastores can live in **one region**
- Workspace controls compute; UC controls data
- Data access = Workspace access + UC permissions
- UC catalog objects are global across attached workspaces
- Cross-metastore queries are not allowed
- Workspace Admin is not a Unity Catalog Admin

UC-Metastore-DEV
- Workspace-DEV
- Workspace-POC

UC-Metastore-UAT
- Workspace-UAT

UC-Metastore-PROD
- Workspace-PROD

- **Workspace = Development + Compute environment.**
- **Unity Catalog = Centralized data governance and security layer.**

Databricks Identities/Principals

In Databricks, **principals** are the *identities* that can be assigned permissions to access data, compute, and workspace resources.

Users

A *user* is an individual person with a Databricks login.

A user has:

- Access to the workspace
- Ability to run clusters (if allowed)
- Ability to access repos
- Access to catalogs, schemas, and tables (if granted)

Service Principals

A *service principal* is a **non-human identity** used by:

- CI/CD pipelines
- Data pipelines
- Azure DevOps / GitHub Actions
- Applications accessing Databricks APIs

Groups

A **group** is a collection of users (and sometimes service principals).

Groups are the **primary way to control permissions** in Databricks.

Groups are used for:

- Workspace access
- Cluster policies
- Catalog permissions (Unity Catalog)
- Workflow/Jobs permissions
- Table read/write permissions
- Repo access permissions

Roles and Permissions in Databricks

Databricks roles fall into **three layers**:

1. Account-Level Roles (Top Level)
2. Workspace-Level Roles
3. Unity Catalog Roles

Account-Level Roles

(Managed in *Databricks Account Console*)

- Account Admin
    - Highest privilege
    - Full control across the **entire Databricks account**

Roles and Permissions in Databricks

Workspace-Level Roles (Managed inside a single workspace)

Access Type / Entitlement	Primary Function	Typical User Persona	Default Status (Non-Admin)
Workspace Admin	Full management of a single workspace (users, settings, resources).	Platform Engineers, Databricks Admins	Not granted
Workspace Access	Access to Data Science & Engineering, and ML features (notebooks, jobs).	Data Scientists, Data Engineers	Granted by default
Databricks SQL Access	Access to Databricks SQL environment (queries, dashboards, SQL warehouses).	Data Analysts, BI Users	Granted by default
Consumer Access	Simplified, read-only access for consuming shared assets (dashboards).	Business Stakeholders, Executives	Not granted (must be explicitly assigned)
Unrestricted Cluster Creation	Ability to create clusters with any configuration (no policy required).	Workspace Admins, Platform Admins	Not granted (must be explicitly assigned)

Roles and Permissions in Databricks

Unity Catalog Roles

Account Admin (Highest Level):

- Manages the entire Databricks account (users, groups, service principals).
- Can create and manage **Unity Catalog Metastores** and link them to workspaces.
- Can assign the **Metastore Admin** role.

Metastore Admin (UC Central Control):

- Has full control over the **Unity Catalog Metastore**.
- Can manage **Storage Credentials** and **External Locations**.
- Can grant top-level privileges like CREATE CATALOG on the metastore.
- Has the ability to read and write all data governed by the metastore (if they grant themselves the necessary privileges).

Workspace Admin (Workspace Control):

- Manages users, clusters, and jobs within a **single workspace**.
- In an auto-enabled UC workspace, they are granted default privileges

  on the attached workspace catalog (e.g., CREATE EXTERNAL LOCATION, ownership of the workspace catalog).

Roles and Permissions in Databricks

Level	Object	Description	Key Privileges
Metastore	METASTORE	The top-level container for all metadata, centrally managed across the account.	CREATE CATALOG, CREATE EXTERNAL LOCATION, CREATE STORAGE CREDENTIAL
1st Level	CATALOG	The first layer, typically used to organize data by organizational unit, environment (e.g., dev, prod), or team.	USE CATALOG, CREATE SCHEMA, SELECT (inherited)
2nd Level	SCHEMA	(Also called Database) The second layer, used to organize related data assets within a catalog.	USE SCHEMA, CREATE TABLE, CREATE VOLUME, SELECT (inherited)
3rd Level	TABLE / VIEW / VOLUME / FUNCTION / MODEL	The lowest level, representing the actual data or compute logic.	SELECT, MODIFY, READ VOLUME, WRITE VOLUME, EXECUTE (on functions/models)

Row-Level & Column-Level Security in Unity Catalog

Unity Catalog provides **fine-grained access control** for data stored in tables and views inside catalogs and schemas.

Two main techniques:

- **Row-Level Security → FILTER POLICIES**
- **Column-Level Security → MASKING POLICIES**

Both are governed and stored **centrally in Unity Catalog**, not per workspace.

Row-Level Security (RLS) using FILTER POLICIES

RLS ensures **different users see different rows** from the same table.

Steps to Implement Row-Level Security:

1. **Policy Function**: A SQL UDF is created that takes table columns as input and returns TRUE/FALSE for row visibility.
2. **Filtering Logic**: The UDF checks user identity or group membership to determine if the user is allowed to see the row.
3. **Applying the Policy**: The UDF is attached to a table as a Row Filter.
4. **Enforcement**: During queries, Databricks runs the UDF on every row and returns only those rows where it evaluates to TRUE.

Column-Level Masking

CLS ensures **sensitive columns are hidden, hashed, or masked** depending on the user role.

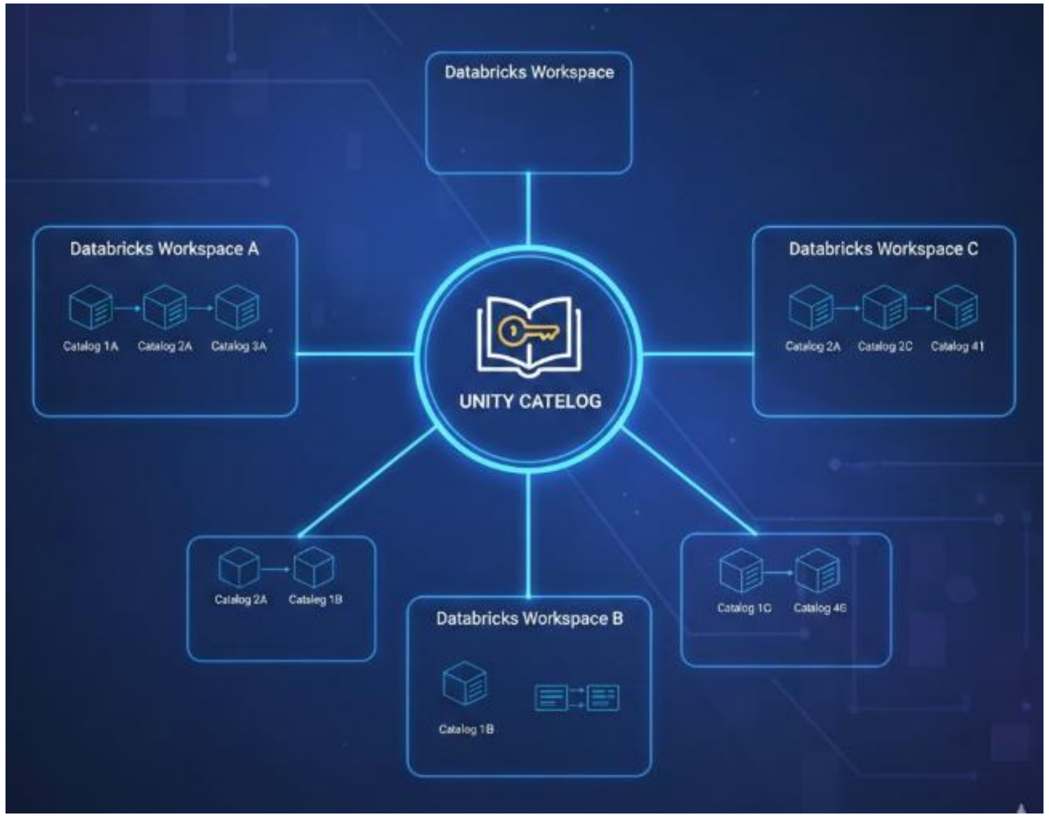
Steps to Implement Column-Level Masking

1. **Identify sensitive columns** that need masking (e.g., age, salary, email).
2. **Create a masking function (UDF)** contains the actual logic for who sees what
3. **Attach the masking function** to a specific column using ALTER TABLE … ALTER COLUMN … SET MASK.
4. **Verify policy behavior** by querying the column as different users or groups.
5. **Update or replace the policy** if masking rules need changes (no table recreation required).
6. **Remove the policy** with DROP MASKING POLICY or ALTER COLUMN … DROP MASK when no longer needed.

Catalog Binding in Workspace

Workspace Catalog Binding is a *Unity Catalog (UC) governance feature* that allows to control **which workspaces can access which catalogs**.

It as a **mapping layer** between a workspace and a catalog to ensure that only authorized workspaces can see or use the data in that catalog.



Workspace Catalog Binding ensures:

- Data isolation across Dev/QA/Prod
- Workspace-level visibility control for catalogs
- Stronger governance under Unity Catalog
- Multi-workspace enterprise setups with controlled catalog access

Delta Sharing in Databricks

Delta Sharing is an **open, secure data sharing protocol** that allows you to **share live data** (tables, views, volumes, and notebooks) with consumers **inside or outside the organization -** without copying data.

It does NOT require:

- The receiver to be on Databricks
- Copying data to another cloud
- Complex pipelines

It is **built on top of Unity Catalog**, ensuring governance and auditing.

Limitations

- Read-Only Access Model
- Only Delta tables are supported for sharing
- UDF functions are not supported

Key Concepts in Delta Sharing

Concept	Meaning
Provider	The owner of the data
Recipient	The user or organization receiving the data
Share	Logical container of tables you want to share
Data Objects	Tables, views, schemas shared
Activation Link / Token	Secure way to deliver access to the recipient

Storage Credentials

- **Definition:**

  A Unity Catalog securable object that stores authentication details required for Databricks to access external data source.
- **Purpose:**
- Separates authentication from external storage for better governance.
- Prevents credentials from being exposed in notebooks, table definitions, or code.

What It Represents:

- The *identity* Databricks uses to prove it is authorized to read/write data in your cloud account.

Security Requirements:

- Should be created and managed only by **metastore admins** or delegated privileged users.
- Protects long-term secrets and enforces centralized credential governance.

External Location

Definition:

- A Unity Catalog object that maps a *cloud storage path* (e.g., ADLS/S3/GCS folder) to a *Storage Credential*.

Purpose:

- Centralizes control over which Databricks users can read/write files in a specific storage location.
- Decouples *storage authentication* (credential) from *storage path* (URL).

What It Represents:

- A governed "doorway" to a directory in external cloud storage.
- Ensures secure access to non-Databricks-managed data locations.

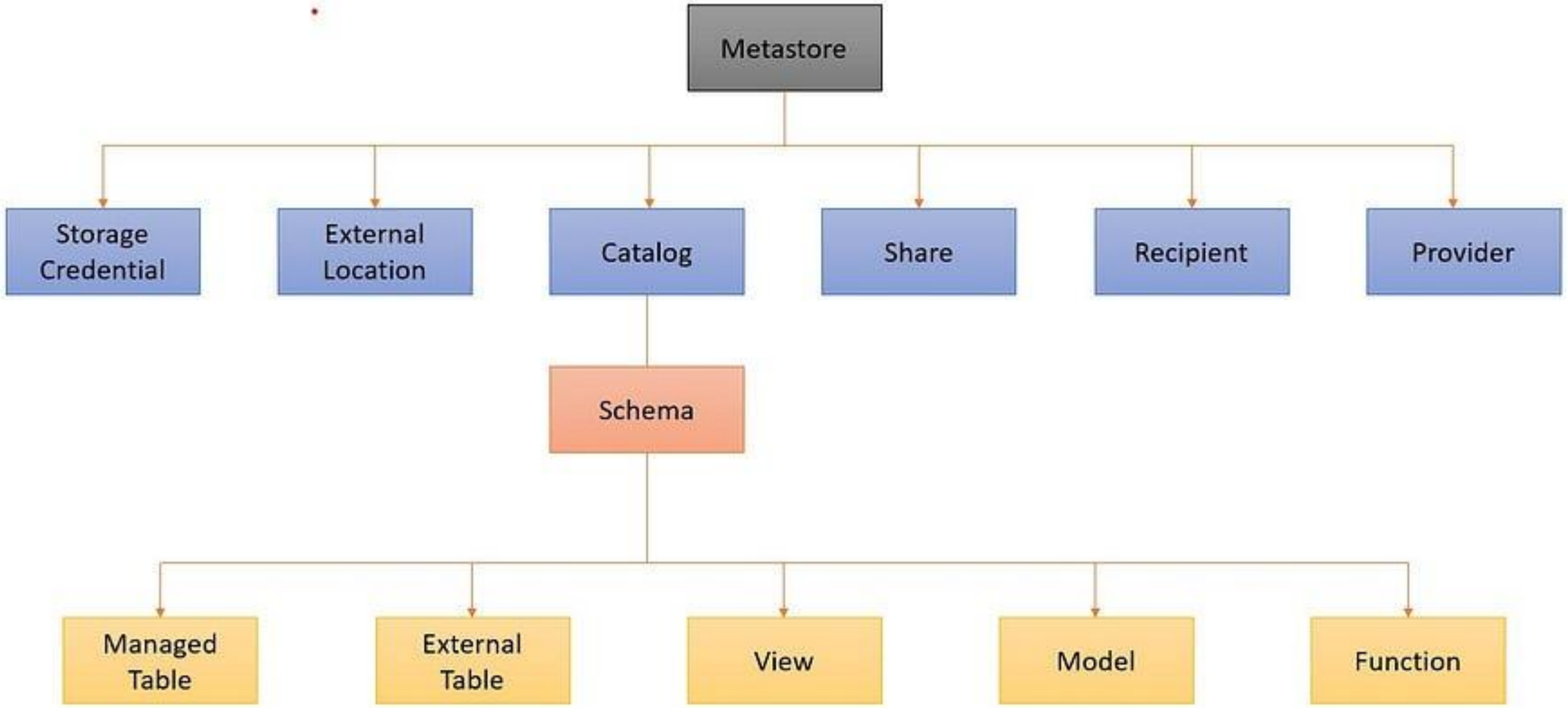
Key Information Stored:

- The **URL** of the storage container/folder (e.g., abfss://raw@storage.dfs.core.windows.net/).
- The **Storage Credential** used to authenticate access.

Common Use Cases:

- Creating external tables on ADLS/S3/GCS.
- Reading/writing raw data in data lakes.
- Hosting external volumes for data engineering workloads.

External Table



External Table

Feature	Managed Table (Unity Catalog)	External Table (Unity Catalog)
Data File Location	Stored in a location determined and fully managed by Unity Catalog	Stored in a location (cloud storage path) explicitly specified by the user using a LOCATION clause.
Data File Management	**Full Management:** Databricks manages the data file lifecycle, including optimization, file size compaction, and automatic deletion.	**Partial Management:** Databricks only manages the metadata. You are responsible for managing the underlying data files.
DROP TABLE Command	**Deletes both:** When you run DROP TABLE, Unity Catalog deletes **both the metadata and the underlying data files** (after an 8-day soft delete period).	**Deletes Metadata Only:** When you run DROP TABLE, Unity Catalog deletes **only the table metadata**. The underlying data files remain in your cloud storage.
Supported Formats	**Delta Lake** (default and recommended) and **Iceberg**.	Supports multiple formats: **Delta Lake**, **Parquet**, **CSV**, **JSON**, **ORC**, etc.
Optimization Features	Benefits from **Predictive Optimization**, automatic OPTIMIZE, VACUUM, and faster metadata reads (metadata caching).	Requires manual execution of OPTIMIZE and VACUUM commands.

External Table

Step 1: Create Storage Credential

```
CREATE STORAGE CREDENTIAL adls_cred
USING MANAGED IDENTITY 'a1b2c3d4-e5f6-7890-g1h2-i3j4k5l6m7n8';
```

- **Step 2: Create External Location**

```
CREATE EXTERNAL LOCATION ext_customer
URL 'abfss://customer-data@customerinfo.dfs.core.windows.net/'
WITH STORAGE CREDENTIAL adls_cred;
```

- **Step 3: Create External Table**

```
CREATE TABLE main.default.my_external_delta_table ( id INT, name STRING, event_date DATE )
USING DELTA LOCATION abfss://customer-data@customerinfo.dfs.core.windows.net/sales/';
```

Connections

A **Connection** defines:

- **How** Databricks authenticates to an external system
- **Where** the external system is (host, port, database)
- **Which credentials** to use (stored securely)

It **does NOT store data** — it only stores **connection metadata**.

Why Connections Are Needed

- Avoid hard-coding usernames/passwords in notebooks
- Enable **centralized governance** via Unity Catalog
- Allow **multiple users and tables** to reuse the same connection
- Support **SQL-based external access**

Database Connections

Used to connect to external RDBMS systems.

Examples:

- MySQL
- PostgreSQL
- SQL Server
- Oracle
- Snowflake

Use cases:

- Query external tables
- Create **Foreign Catalogs**
- Read/write data via Databricks SQL

Foreign Catalog

A **Foreign Catalog** is a **Unity Catalog object** that allows **Databricks to reference and query metadata that lives outside Databricks**, in an **external metastore or database**, **without copying the data**.

It acts as a **logical bridge** between Databricks and an **external system's catalog**.

- Databricks does **not own the data**
- Databricks **does not manage the lifecycle**
- Only **metadata pointers** are registered

Foreign catalogs are commonly used with:

- External relational databases (MySQL, PostgreSQL)
- External Hive metastores
- Federated query sources

Aspect	**Foreign Catalog**	**External Catalog**
Data location	External database	External cloud storage
Primary Use	Cross-platform querying	Storing large-scale Delta/Parquet files
Storage type	RDBMS / external metastore	ADLS / S3 / GCS
Access method	Federation / JDBC	File-based

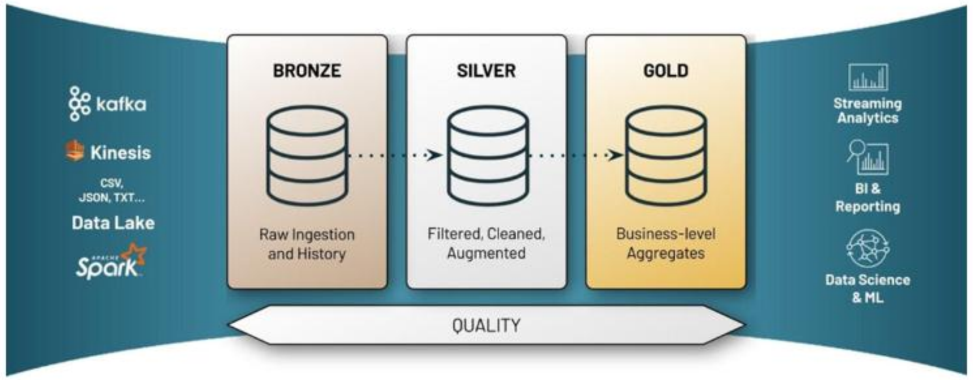
Medallion Architecture

The **Medallion Architecture** is a **layered data design pattern** used in the **Databricks** and modern data lakehouses.

It organizes data as it moves from **raw → cleaned → business-ready**, improving **data quality, governance, and performance**.

Key Benefits

- **Improved data quality** by progressively cleaning and validating data across layers
- **Clear separation of concerns** between raw ingestion, transformation, and analytics
- **Better performance** by serving pre-aggregated, optimized Gold tables
- **Scalability** to handle large batch and streaming workloads independently per layer
- **Full data lineage and auditability** from business metrics back to raw source data
- **Simpler debugging and reprocessing** by replaying data from Bronze or Silver layers
- **Schema evolution support** without breaking downstream consumers
- **Reusability of datasets** across multiple analytics, BI, and ML use cases
- **Cost optimization** by isolating heavy transformations and reducing repeated compute
- **Stronger governance and security** through controlled access at each layer



Medallion Architecture – 3 Layers

Name	Purpose	Data Characteristics	Typical Operations
Bronze (Raw)	Capture raw data exactly as received	• Append-only • Immutable • Schema-on-read • High volume	• Ingestion only • Minimal parsing • No deduplication
Silver (Cleaned)	Clean, validate, and enrich data	• Deduplicated • Typed schema • Conformed data	• Data cleansing • Deduplication • Joins & lookups • SCD handling
Gold (Curated)	Business-ready analytics & KPIs	• Aggregated • Optimized • Stable schema	• Aggregations • KPI calculations • Dimensional modeling

Traditional vs Medallion

Feature	Traditional Data Warehouse	Medallion (Lakehouse)
Approach	**ETL** (Transform before loading)	**ELT** (Load raw, transform after)
Data Types	Primarily structured (SQL tables)	Structured, semi-structured, & unstructured
Flexibility	**Rigid**: Schema must be defined upfront	**Flexible**: Schema can evolve over time
Storage Cost	High (expensive proprietary storage)	Low (cheap cloud object storage/S3/ADLS)
Primary Goal	Business Intelligence & Reporting	BI, Data Science, and Machine Learning
Use cases	• Pure BI & reporting • Stable schemas • Strong SQL-only workloads • Regulated legacy environments	• Real-time or near-real-time data • Streaming + batch together • ML / AI use cases • Cloud-first modernization • Cost optimization is important

SCD-2

In **Databricks**, **Slowly Changing Dimension (SCD) Type 2** is used to **preserve full history of changes** in dimensional data (customers, products, policies, etc.) by **creating a new row for each change** instead of overwriting existing data.

Every change creates a new versioned row; old rows are expired, not deleted.

Id	Name	effective_date	end_date
1	India	2025-12-15	Null
2	United Kingdom	2025-12-15	Null
3	Canada	2025-12-15	Null
4	Australia	2025-12-15	Null
5	Germany	2025-12-15	Null
6	France	2025-12-15	Null
7	Japan	2025-12-15	Null
8	Brazil	2025-12-15	Null
9	South Africa	2025-12-15	Null
10	United States	2025-12-15	Null

Attribute Change & New Row

Id	Name
10	USA
11	Ireland

	Id	Name	effective_date	end_date
	1	India	2025-12-15	Null
	2	United Kingdom	2025-12-15	Null
	3	Canada	2025-12-15	Null
	4	Australia	2025-12-15	Null
	5	Germany	2025-12-15	Null
	6	France	2025-12-15	Null
	7	Japan	2025-12-15	Null
	8	Brazil	2025-12-15	Null
	9	South Africa	2025-12-15	Null
Old Row	10	United States	2025-12-15	2025-12-17
New Row	11	Ireland	2025-12-15	Null
Active Row	10	USA	2025-12-17	Null

Databricks CLI

The **Databricks CLI** is a **command-line tool** that lets you **interact with a Databricks workspace programmatically** from your local machine or CI/CD pipeline—without using the UI.

It is commonly used for:

- Workspace automation
- Job deployment
- Cluster management
- DBFS operations
- CI/CD integration (DevOps)

Installation Steps

1. Open PowerShell/Terminal as Administrator
2. Install Databricks CLI

   winget install Databricks.DatabricksCLI (Windows)

   brew install databricks (Mac)

3. Restart Terminal and verify Installation

   databricks -v

4. Start Login Process

   databricks auth login --host <workspace-url>

5. Complete Browser Authentication
6. Test the Connection

   databricks clusters list

Databricks CLI Commands

Modules	Command	Description
Authentication		
	databricks auth login --host <databricks url>	Authenticate using OAuth (standard for modern accounts).
	databricks auth profiles	List all configured connection profiles.
	databricks auth describe	Check current auth
	databricks version	Check the current version of the CLI.
Filesystem & Workspace		
	databricks fs ls dbfs:/	List files in a root DBFS path.
	databricks fs ls dbfs:/Volumes/inceptez_catalog/inputdb/customerdata	List files in a specific DBFS path.
	databricks fs mkdir dbfs:/Volumes/inceptez_catalog/inputdb/customerdata/test	Create a new directory in DBFS.
	databricks fs cp D:/Training/data/emp_perf.csv dbfs:/Volumes/inceptez_catalog/inputdb/customerdata/test/	Copy a file from local to Databricks.
	databricks fs rm -r dbfs:/tmp/old_data	Delete a file or directory (recursively).
	databricks workspace list /Users/azurede007@gmail.com	List the files in the workspace
	databricks workspace export "/Users/azurede007@gmail.com/Lab01-Basics" --file "./Lab01-Basics.py"	Download notebook from the workspace to local
	databricks workspace import "./Lab01-Basics.py" "/Users/azurede007@gmail.com/Lab01-New" --language PYTHON --format SOURCE --overwrite	Upload a local folder of notebooks to the workspace.

Databricks CLI Commands

Modules	Command	Description
Unity Catalog		
	databricks catalogs list	Show all available Unity Catalog catalogs.
	databricks schemas list <catalog-name>	List schemas (databases) within a catalog.
	databricks tables list <catalog-name> <schema-name>	List all tables, including your GCS external tables.
	databricks tables get <catalog-name>.<schema-name>.<table-name>	List all the columns in the table
	databricks storage-credentials list	View all cloud credentials (AWS/GCP/Azure).
	databricks external-locations list	List all GCS/S3/ADLS external paths.
Clusters & Jobs		
	databricks clusters list	List all clusters and their status (Running/Terminated).
	databricks clusters start --cluster-id <id>	Spin up a specific cluster.
	databricks jobs list	List all scheduled workflows.
	databricks jobs run-now --job-id <id>	Trigger a job immediately.
Secrets Management		
	databricks secrets create-scope <scope-name>	Create a new logical container for secrets.
	databricks secrets put-secret <scope> <key>	Opens an editor to paste your secret value.
	databricks secrets list-scopes	See all available secret scopes.
	databricks secrets list-secrets <scope>	List keys within a specific scope.

DBSecrets in Databricks

Databricks Secrets provide a secure way to store sensitive information like API keys, database passwords, and connection strings.

Instead of hardcoding these in notebooks, store them in **Secret Scopes** and retrieve them using the dbutils.secrets utility

Core Concepts

- **Secret Scope:** A logical container (like a folder) used to group related secrets.
- **Secret:** A key-value pair where the value is the sensitive data.
- **Access Control List (ACL):** Defines who can READ, WRITE, or MANAGE secrets within a specific scope.

Managing Secrets

- Secret creation and management (scopes and values) is primarily handled through the Databricks CLI, as the UI intentionally limits direct secret creation for security reasons.

- From UI, allowed to create only Secret Scope but not Secret

  https://<databricks-workspace-url>#secrets/createScope

- dbutils usage is restricted to listing and reading secrets, while creating scopes or storing secret values is deliberately allowed only via the Databricks CLI or UI to ensure security.

DBSecrets in Databricks

Common CLI Commands:

- **Create a scope:** `databricks secrets create-scope <scope-name>`
- **Add a secret:** `databricks secrets put-secret <scope-name> <key-name>`
- **List scopes:** `databricks secrets list-scopes`

Use Secrets in a Notebook

Once a secret is stored, can retrieve it using the `dbutils` library.

Databricks automatically **redacts** (hides) the output if you try to print it.

```
#List the Secrets
dbutils.secrets.listScopes()
dbutils.secrets.list("app-auth-secrets")

# Retrieve the secret
db_user = dbutils.secrets.get(scope="app-auth-secrets", key="app_user")
db_password = dbutils.secrets.get(scope="app-auth-secrets", key="app_password")

# Use it in a connection string (it will appear as [REDACTED] if printed)
jdbc_url = f"jdbc:mysql://host:3306/db?user={db_user}&password={db_password}"
```

Azure Key Vault and DBSecrets in Databricks

Integrating **Azure Key Vault (AKV)** with **Azure Databricks** is the industry-standard way to manage sensitive credentials - like database passwords, API keys, and storage connection strings—without hardcoding them into your code.

1. Prepare Azure Key Vault

- **Permissions:** Ensure your user account has the **Key Vault Administrator** or **Contributor** role on the Key Vault to create the link.
- **Get Vault Details:** Go to your Key Vault in the Azure Portal > **Properties**.
- **Copy & Save:**
    - **Vault URI** (this is the DNS Name).
    - **Resource ID**

2. Create the Secret Scope in Databricks

There is no "button" in the Databricks UI for this; you must use a specific URL:

- **The URL:** Open your Databricks workspace and append `#secrets/createScope` to the end of the URL.
    - *Example:* `https://adb-12345.6.azuredatabricks.net/?o=123#secrets/createScope`
    - **Note:** The "S" in `createScope` must be **uppercase**.
- **Fill the Form:**
    - **Scope Name:** A name for your scope (e.g., `kv-storage-scope`).
    - **DNS Name:** Paste the **Vault URI**.
    - **Resource ID:** Paste the **Resource ID**.
- **Manage Principal:** Choose **Creator** (to limit access) or **All Users** (if everyone in the workspace should use these secrets).

3. Retrieve Secrets in a Notebook

Once linked, use the `dbutils.secrets` utility. The values will always be masked as `[REDACTED]` in the output for security.

Access Azure Storage using Service Prinicipal in Databricks

1. Register an Identity (Entra ID)

- Go to **Microsoft Entra ID** > **App Registrations** > **New Registration**.
- **Save these:** Application (Client) ID and Directory (Tenant) ID.
- Go to **Certificates & Secrets** > **New Client Secret**. Copy the **Value** immediately.

2. Set Permissions (Storage Account)

- Go to your **Storage Account** > **Access Control (IAM)**.
- **Add Role Assignment:** Assign the role **Storage Blob Data Contributor** to the App Registration you just created.

3. Connect in Databricks

- Use dbutils to pull the credentials securely into your notebook

Databricks Compute

In Databricks, **Compute** refers to the infrastructure (CPU, memory, and storage) used to run data processing, machine learning, and analytics workloads. Think of it as the "engine" that executes your code (Python, SQL, Scala, or R).

It powers everything do in a Databricks workspace: notebooks, jobs, SQL queries, Streaming pipelines, ML training and more

Databricks has evolved to offer three main categories of compute, each designed for a specific type of work.

1. **Databricks serverless compute** – fully managed compute where databricks automatically provisions, scale, patches and isolates the underlying resources on demand.
2. **All-Purpose Compute** – Consists of persistent, user-managed clusters that remain running until stopped, support multiple users, and are optimized for interactive notebook development, debugging, and exploratory analytics
3. **Job Compute** – ephemeral, auto-provisioned cluster type optimized for running automated tasks and pipelines to ensure cost-efficient execution

Databricks Compute – Serverless Compute

- **Serverless Compute for Notebooks**
  - On-demand scalable compute for interactive notebooks
  - Supports SQL & Python execution
  - Attach notebooks to “Serverless” compute – no manual cluster setup
  - Automate VM provisioning, security, scaling & teardown handled by notebooks
- **Serverless Compute for Jobs**
  - On-demand scalable compute for Databricks pipeline tasks
  - No Infrastructure setup – attach tasks directly to “Serverless” compute
  - Automatic resource allocation & Scaling per task
  - Per-seconds billing based on actual execution time
- **Serverless Compute for SQL Warehouses**
  - Elastic, fully managed compute endpoints for SQL analytics, dashboards & ad-hoc queries
  - Use in SQL editor or notebooks without manual cluster setup
  - Auto-scales to meet concurrency demands

Databricks Compute - All Purpose Compute

These are the traditional "clusters" that you create, configure, and manage manually. They run in your own cloud account (AWS/Azure/GCP).

Designed for collaboration. Multiple people can attach their notebooks to one cluster.
We can manually start, stop, and restart them.

Key Characteristics:

- Manually created, persistent clusters that run until stopped
- Ideal for interactive notebooks, debugging, and data experimentation
- Supports multiple users attaching concurrently
- Configurable auto-scaling & auto-termination to manage size and costs
- Best for ad-hoc development & exploration, not for production pipelines or scheduled jobs

Databricks Compute - Job Compute

Dedicated to a single automated task (a "Job"). Databricks creates them when the job starts and destroys them as soon as it ends. This is significantly cheaper than using All-Purpose clusters for jobs.

Key Characteristics:

- Ephemeral spin up for automated tasks (jobs, pipelines, scripts)
- Spin-up and tear-down design: Starts with the job and terminates on completion
- Cost efficient billing – pay only for actual runtime
- Optimized for automation & performance, not interactively or multi-user use
- Stateless environment ensures consistency and clean starts for production workflows

Databricks Compute

Feature	Serverless (General)	Classic (All-Purpose)	Classic (Jobs)	SQL Warehouses
Primary Use	Ad-hoc Python/SQL	Interactive Dev/ML	Production ETL	BI / SQL Analytics
Startup Time	Seconds (~20–30s)	5–10 Minutes	5–10 Minutes	Seconds (Serverless)
Engine	Standard Spark	Standard Spark	Standard Spark	Photon (Optimized C++)
Scaling	Fully Automatic	User-managed	User-managed	Intelligent Scaling
Language	Python, SQL, R, Scala	All Languages	All Languages	SQL Only
Cost (DBU)	High (Bundled)	Medium-High	Lowest	Medium (Optimized)

Databricks Compute – Key Concepts

Key Concepts to Know

- **Photon:** An optimized engine written in C++ that speeds up Spark and SQL workloads. It is standard on SQL Warehouses and an option for most clusters.
- **Autoscaling:** A feature where Databricks adds or removes "Worker" nodes based on how much data you are processing, helping you save money during quiet periods.
- **Instance Pools:** Pre-provisioned collection of idle virtual machines that clusters can draw from to start up rapidly, reducing provisioning time and improving resource efficiency. **Cloud VM Cost** (No DBU)
- **DBR's:** Pre-configured execution environment used by Databricks compute (clusters and SQL warehouses) to run workloads. It bundles Apache Spark, system libraries, language runtimes, and Databricks optimizations into a single, versioned package.

Databricks DBU’s

In **Databricks**, a **DBU (Databricks Unit)** is a **billing unit** that measures how much **compute power** you use.

Think of a **DBU like “electricity units”** — the more compute you run and the longer you run it, the more DBUs you consume.

How is a DBU Calculated?

The calculation is basically a multiplication of three things:

Total DBUs = (Number of Nodes) X (DBU Rate per Node) X (Time Used)

- **Number of Nodes:** How many computers (VMs) are in your cluster.
- **DBU Rate:** This is a fixed value assigned to a specific computer type. A “high" computer with 64GB of RAM will have a higher DBU rate (e.g., 4.0) than a small one with 8GB of RAM (e.g., 0.5).
- **Time Used:** Databricks bills you by the **second**. If you run a cluster for 10 minutes, you only pay for 10 minutes of DBUs.

Databricks DBU's Calculation – Use Case

Run a 4-node Jobs cluster for 1 hour on a Premium tier where node type is Standard_F4

Component	**Metric for Standard_F4**
Nodes	4 Workers (plus 1 Driver node usually required, total 5)
DBU Count	~0.50 to 0.75 DBUs per hour (per node)
DBU Price	~₹25 per DBU (Jobs Compute - Premium Tier)
Azure VM Price	~₹15 to ₹22 per hour (per Standard_F4 node)

Step-by-Step Cost Calculation (1 Hour Run)

A. Databricks DBU Cost

For **Jobs Compute** on the **Premium Tier**, the DBU rate is discounted compared to All-Purpose compute

- **Nodes:** 5 nodes (4 Workers + 1 Driver)
- **DBU Rate:** ~0.75 DBUs per F4 instance per hour.
- **Total DBUs 5** nodes x 0.75 DBUs = 3.75 DBUs
- **Cost in INR:** 3.75 DBUs x ₹25/DBU = ₹93.75

Databricks DBU’s Calculation – Use Case

B. Azure Infrastructure Cost (VMs)

The Standard_F4 is a compute-optimized instance (4 vCPUs, 8GB RAM).

- **Nodes:** 5 nodes
- **VM Rate (India):** ~₹18.50 per hour per node.
- **Total VM Cost:** 5 x ₹18.50 = ₹92.50

3. Total Estimated Cost

Expense Category	**Estimated Cost (INR)**
Databricks Platform (DBUs)	₹93.75
Azure Infrastructure (VMs)	₹92.50
Total per Hour	**₹186.25**

Databricks Policy

In **Databricks**, a **compute policy** (also called a **cluster policy**) is a **set of governance rules** that control **how compute resources (clusters and warehouses) can be created and used**.

In simple terms, **a compute policy defines what users are allowed to configure when they create compute**.

Category	Type	Who is it for?	Main Characteristic
Unrestricted	Policy	Admins / Power Users	No limits; can change any setting (RAM, CPU, Workers).
Personal	Policy	Individuals	**Single-node** (no workers); cheapest and fastest for small tasks.
Power User	Policy	Data Scientists	Multi-node; supports large datasets and ML runtimes.
Shared	Access Mode	Teams / BI	**Multi-user** isolation; many people use one cluster securely.
Legacy	Feature/Mode	Old Workflows	Refers to old Table ACLs or Hive Metastore (non-Unity Catalog).

Custom Policy in Databricks

A **cluster policy** in **Databricks** is a set of rules that **controls how clusters are created**, such as:

- Allowed node types
- Minimum / maximum workers
- Whether users can change certain settings

This helps with **cost control, security, and standardization**.

Policy Field Types

Type	Meaning
fixed	User cannot change the value
range	User can choose within limits
allowlist	User can pick from allowed values
forbidden	Setting is completely blocked

To create a policy, navigate to **Compute** > **Policies** > **Create policy**. We can define policies using the UI or by writing a JSON definition.