Unity Catalog

Name: Priyeshwar

Mail: priyesh2664@gmail.com

Introduction

Databricks Unity Catalog (UC) is a centralized governance solution designed to manage data and AI assets within the Databricks ecosystem. It works seamlessly across multiple clouds and platforms, providing a unified approach to securing, organizing, and tracking data throughout its lifecycle. UC handles both structured and unstructured data, as well as machine learning models, notebooks, and dashboards, all under one governance framework. By offering fine-grained access control and integrated data lineage, it helps organizations meet compliance requirements while enabling secure collaboration.

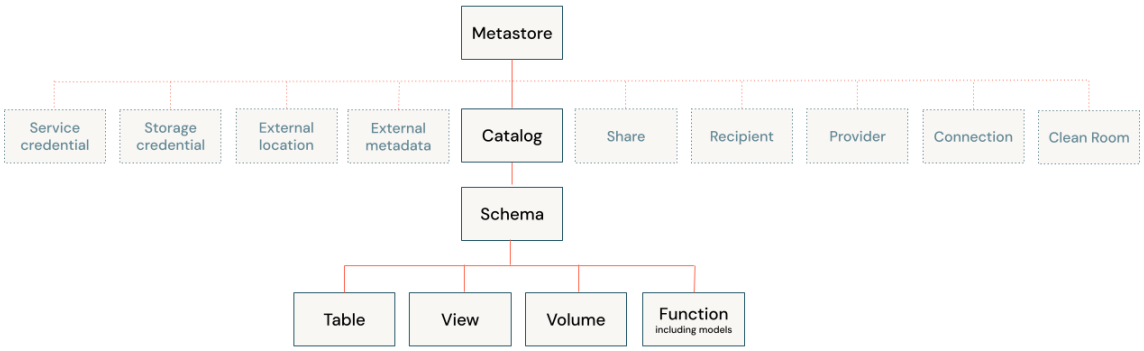
Purpose & Benefits

Unity Catalog addresses the challenges of fragmented data governance by creating a single control point for managing access policies across workspaces and data sources. It allows administrators to define permissions once and enforce them consistently, avoiding the complexity of maintaining separate tools or rules. This reduces security risks and ensures compliance with data privacy regulations. Additionally, UC enhances productivity by improving data discovery and providing built-in auditing and lineage features. Teams can quickly find the data they need, understand its origin, and see how it has been transformed or used. For organizations sharing data across departments or even external partners, Delta Sharing makes it possible to do so securely without compromising governance.

Architecture & Components

Unity Catalog is built around a clear hierarchical object model:

- Metastore – The top-level container that organizes metadata for all data assets.
- Catalog – A logical grouping of schemas, often aligned with a department, data domain, or environment (e.g., production vs. development).
- Schema – Also called a database, it contains tables, views, and volumes.
- Tables, Views, and Volumes – Tables hold structured data, views are virtual tables created with queries, and volumes store unstructured files.
- Models – Registered machine learning models that can be governed alongside other data assets.



On top of this structure, UC provides access control, which uses an inheritance model where permissions granted at a higher level flow down to lower-level objects. It also includes operational intelligence tools that give real-time visibility into how assets are being used, with AI-powered alerts and monitoring capabilities.

Compute Modes & Supported Formats

Unity Catalog supports different cluster access modes to meet varying needs:

- Shared Access Mode – Suitable for collaborative environments where multiple users share the same cluster, balancing isolation and efficiency.
- Single User Mode – Provides maximum isolation for workloads run by one user, commonly used for automated jobs or machine learning experiments.
- No-Isolation Shared Mode – A legacy mode that is not supported because it does not meet UC's security requirements.

For data formats, UC enforces Delta format for all managed tables, ensuring high performance, ACID compliance, and compatibility with governance features. External tables can use more formats, including CSV, JSON, Avro, Parquet, ORC, and Text, giving flexibility when integrating with legacy or external systems.

How to create Unity Catalog ?

Prerequisites

Before creating Unity Catalog in Azure, ensure the following:

- An **Azure subscription**.
- An **Azure Databricks Premium workspace**.
- Admin privileges in Databricks.
- An **Azure Active Directory (AAD) identity** with permission to create resources.
- An **Azure Data Lake Storage Gen2 (ADLS)** account with hierarchical namespace enabled.

Steps to Create Unity Catalog in Azure

Step 1: Configure Azure Storage (ADLS Gen2)

- Create an **Azure Data Lake Storage Gen2 account**.

- Enable **hierarchical namespace** for directory-based access.
- Create a **container** in ADLS to store Unity Catalog-managed data.

Step 2: Create a Storage Account & Assign IAM Role

- Go to the ADLS account → **Access Control (IAM)**.
- Assign the following roles to the **Databricks workspace managed identity**:
    - **Storage Blob Data Contributor** (for read/write access).
    - **Storage Blob Data Owner** (for full access if needed).

Step 3: Create Unity Catalog Metastore

1. Log in to the **Azure Databricks account console**.
2. Navigate to **Data → Metastores**.
3. Click **Create Metastore**.
4. Provide:
    - **Name** (e.g., "azure_uc_metastore").
    - **Region** (same as the Databricks workspace).
    - **ADLS storage path** (container created in Step 1).
    - Assign the **Azure managed identity** created earlier.
5. Click **Create**.

Step 4: Assign the Metastore to Workspaces

- In the Metastore view, open the **Workspaces tab**.
- Click **Assign to Workspaces**.
- Select the Azure Databricks workspace(s) where UC should be enabled.

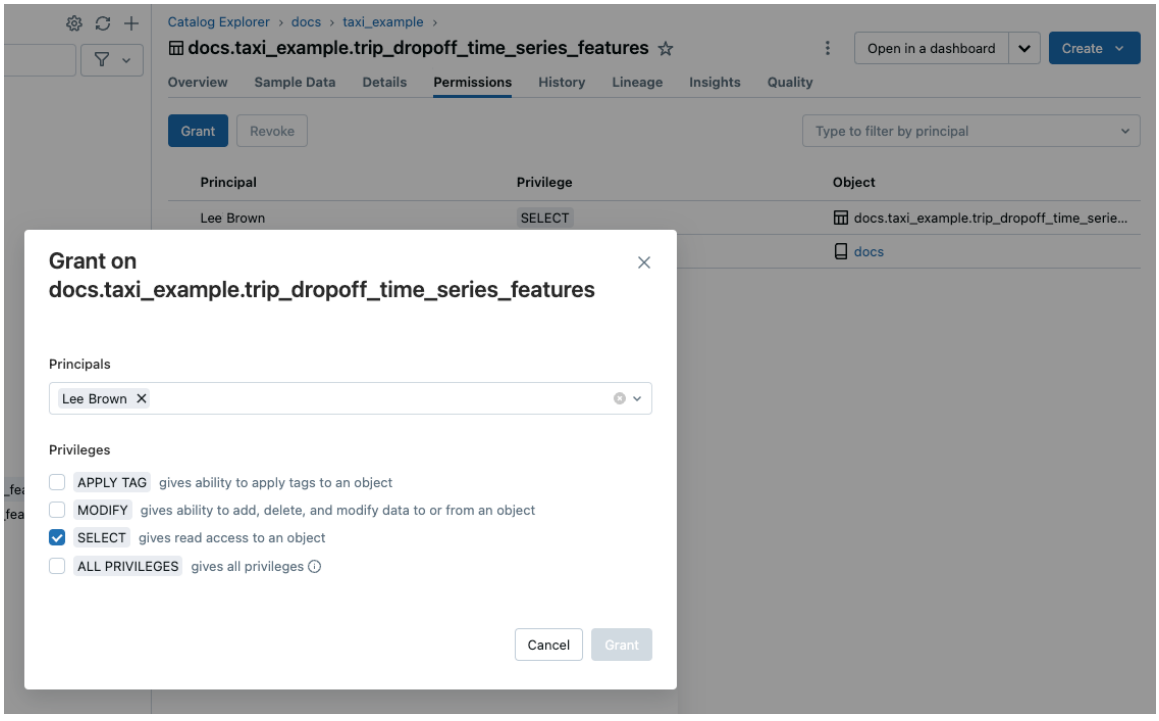
Step 5: Enable Unity Catalog in Databricks Workspace

- Open the workspace.
- Go to **Data tab** → confirm the assigned metastore.
- Test by running:
- SELECT CURRENT_METASTORE();
- If a metastore ID is returned, UC is enabled.

Granting and Revoking Access to Securable Objects

Unity Catalog uses a privilege-based model for governing securable objects such as catalogs, schemas, and tables. Privileges can be granted and revoked at any level, with inheritance ensuring that child objects receive the same access by default.

For example, using SQL:
GRANT CREATE TABLE ON SCHEMA mycatalog.myschema TO `finance-team`;

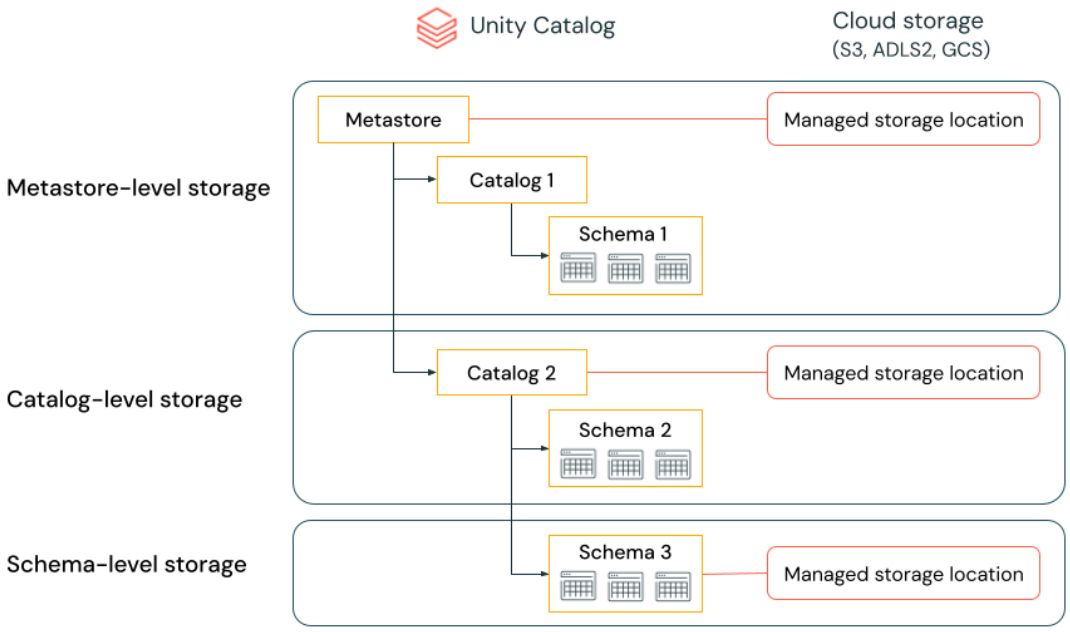


Other methods include Catalog Explorer (UI), the Databricks CLI, and REST APIs.

Managed Storage Location Hierarchy
Unity Catalog provides flexibility in defining managed storage locations depending on organizational requirements for data isolation. Locations can be assigned at the metastore, catalog, or schema level.

Example: A compliance policy requires HR production data to be stored in a secure container:
abfss://mycompany-hr-prod@storage-account.dfs.core.windows.net/unity-catalog

By creating a catalog 'hr_prod' and assigning this location, all managed tables in the catalog are stored in the specified container. Optionally, schema-level locations can be used for finer control.



Evaluation order of storage locations:

1. Schema-level location (highest priority).
2. Catalog-level location.
3. Metastore-level location (default if none defined above).

Key Features in Azure Unity Catalog

- **Centralized Governance**: Policies apply across all Azure Databricks workspaces.
- **Integration with Azure AD**: Access control is based on Azure identities.
- **Data Lineage**: Track data flows across pipelines and notebooks.
- **Cross-Cloud Sharing**: Use Delta Sharing to securely share data across Azure and other clouds.

Best Practices

Unity Catalog is most effective when combined with a strong governance strategy:

- Use catalogs for isolation – Separate data by domain, environment, or sensitivity to maintain clear access boundaries.
- Apply storage isolation – Configure different cloud storage locations for different catalogs or schemas, ensuring physical as well as logical separation.
- Group-based ownership – Assign ownership of objects to groups rather than individuals, making it easier to manage access when personnel changes occur.
- Leverage permission inheritance – Grant permissions at higher levels (catalog or schema) so they apply automatically to lower-level objects, reducing admin work.

- Attribute-based access control – Use dynamic views to filter or mask data based on a user's identity or group membership, enabling fine-grained security.
- Secure clusters – Enforce policies for cluster creation and usage to ensure all compute resources are compatible with UC's security model.
- Comprehensive auditing – Regularly review UC's audit logs to monitor access patterns, detect anomalies, and support compliance reporting.
- Use Delta Sharing for collaboration – Share data securely between teams or organizations without exposing direct storage access.

Limitations

While Unity Catalog provides extensive governance capabilities, it has some constraints to be aware of:

- Older Databricks Runtime versions (below 11.3 LTS) lack full support for UC features, so upgrading is essential.
- Row-level and column-level security via dynamic views is not available for R workloads.
- Bucketing and custom partition schemes are not supported, which can affect certain performance tuning approaches.
- Overwrite mode for DataFrame writes works only with Delta tables, not with other formats.
- Some UDF types (Python or Scala) have runtime-specific support limitations.
- Object naming has restrictions on length, characters, and case sensitivity.
- Workspace-local groups cannot be used in GRANT statements; account-level groups are required.

- Concurrent writes to the same external storage location from multiple metastores can cause consistency issues, though reading is safe.
