**Madiha Aimon Tappal**

[madihaaimon@gmail.com](mailto:madihaaimon@gmail.com)

**Data Engineering Batch – 1**

**Day – 21 Assignment**

**Azure Databricks**

**Unity Catalog in Databricks:**

## Centralized Governance for Your Data

Unity Catalog is a powerful tool in Databricks that offers **centralized access control, auditing, lineage, and data discovery capabilities** across all your workspaces. Think of it as the single source of truth for your data governance needs.

**Key Features:**

* **Define Once, Secure Everywhere:** Set access policies for datasets and manage user permissions in one place, eliminating the need for repetitive configurations across workspaces.
* **Standards-Compliant Security:** Leverage familiar ANSI SQL syntax to grant permissions at various levels (catalogs, databases, tables, views), ensuring consistency and ease of use.
* **Automatic Data Lineage:** Track how data flows through your pipelines, from source to consumption, without manual setup. This transparency empowers data quality management and impact analysis.
* **Centralized Auditing:** Get a comprehensive view of all data access and usage across your workspaces, aiding compliance and security monitoring.
* **Enhanced Data Discovery:** Easily search and explore available datasets across workspaces, accelerating collaboration and knowledge sharing.
* **Unified Governance:** Integrate with existing data catalogs and governance tools, maximizing your investments and simplifying data management.

**Use Cases:**

* **Simplifying Data Access Control:** Streamline user and group management, ensuring consistent access policies across your data lake.
* **Improving Data Quality:** Utilize data lineage to identify issues and dependencies, accelerating root cause analysis and ensuring data reliability.
* **Enhancing Security and Compliance:** Gain centralized visibility into data access and usage, facilitating regulatory compliance and security best practices.
* **Boosting Collaboration:** Empower data sharing and accelerate insights by allowing teams across workspaces to easily discover and access relevant datasets.

**Architectural Approach:**

* **Centralized Metastore:** Instead of individual metastores per workspace, Unity Catalog uses a single, shared repository for data definitions and metadata. This ensures consistency and eliminates siloed information.
* **Distributed Infrastructure:** The metastore itself is geographically distributed across multiple regions, offering high availability and scalability.
* **Microservices Architecture:** Unity Catalog utilizes a microservices architecture, decomposing functionalities into modular services for increased flexibility and resilience.

**Governance Principles:**

* **Role-Based Access Control (RBAC):** Granular control over data access is granted based on defined roles and permissions, ensuring data security and regulatory compliance.
* **Lineage Tracking:** Captures the transformation history of data across various stages, revealing dependencies and facilitating impact analysis.
* **Auditing and Logging:** Detailed records of data access and usage are captured for analysis and compliance purposes.
* **Data Discovery and Search:** Enables users to easily search and browse available datasets across workspaces, fostering collaboration and knowledge sharing.

**Data Lineage Benefits:**

* **Data Quality Management:** Identify data quality issues and root causes by tracing dependencies and transformations.
* **Impact Analysis:** Understand the downstream impact of changes made to data pipelines.
* **Auditability and Compliance:** Demonstrate data provenance and track regulatory compliance adherence.

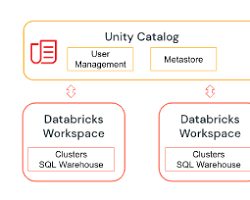
**Security Features:**

* **Encryption:** Data at rest and in transit is encrypted using industry-standard algorithms.
* **Authentication and Authorization:** Secure access to the metastore is enforced through robust authentication and authorization mechanisms.
* **Activity Monitoring:** Real-time monitoring of user activity and data access attempts ensures security and anomaly detection.

**Integration with Existing Solutions:**

* **Open APIs:** Unity Catalog offers open APIs for integrating with existing data catalogs, governance tools, and cloud platforms.
* **Connectors:** Pre-built connectors simplify integration with popular data storage systems and frameworks.

**Architecture diagram of Unity Catalog in Databricks:**

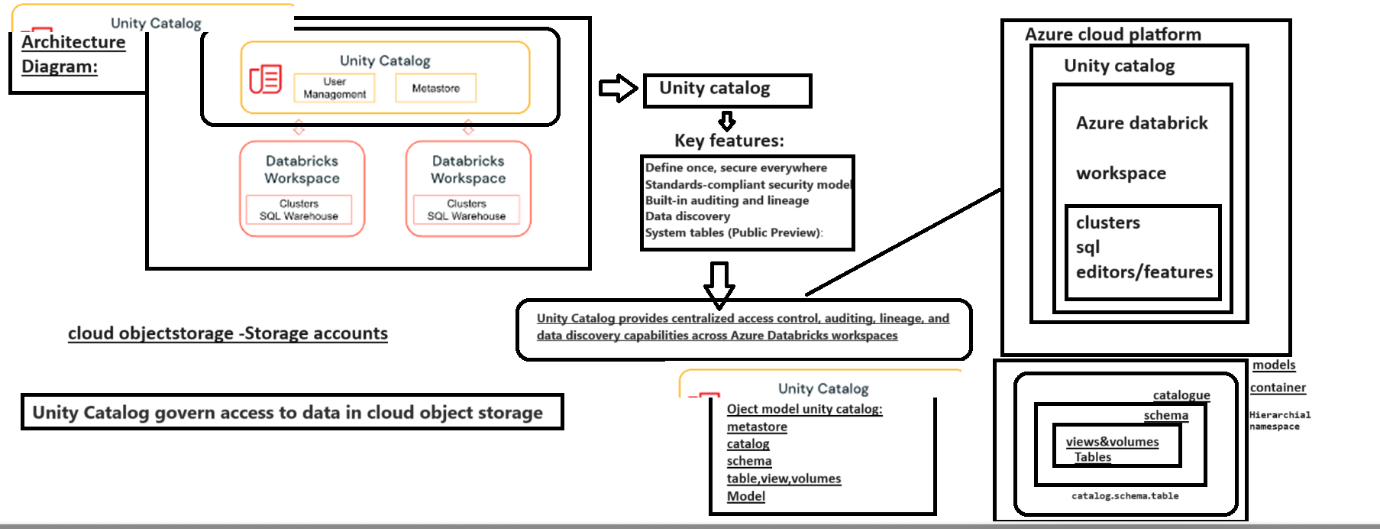


The diagram depicts the key components of Unity Catalog and how they interact with each other. As you can see, the architecture is built around a central metastore that stores all the metadata about your data assets. This metastore is accessed by various clients, such as notebooks, Spark jobs, and SQL queries, through a unified API. The API enforces access control and ensures that only authorized users can access data.

The lineage service tracks the lineage of your data, which means it keeps track of how your data has been transformed from its raw source to its final destination. This information is important for understanding the impact of changes to your data pipelines and for ensuring data quality.

The auditing service logs all access to the metastore and lineage service. This information can be used for security purposes and for compliance with regulations.

The Unity Catalog architecture is designed to be scalable and secure. It can be deployed on-premises or in the cloud, and it can handle large volumes of data.



**Unity Catalog object model in Databricks:**

Unity Catalog uses a hierarchical object model with three primary levels:

1. **Metastore:** The top-level container for metadata, providing a single source of truth for data definitions and access control across your workspaces.
2. **Catalog:** Represents a logical grouping of data assets within a metastore. You can organize catalogs based on project, department, or other criteria.
3. **Schema (Database):** Contains tables, views, and volumes within a catalog. Think of it as a logical container for related data assets.
4. **Table/View/Volume:** The atomic units of data storage.
   * Tables store structured data.
   * Views offer virtual representations of underlying data without physical storage.
   * Volumes manage non-tabular data like images or documents.

This hierarchical structure offers clear organization and granular access control across your data lake in Databricks. With this understanding, you can effectively navigate and manage your data using the Unity Catalog features.

**Central Metadata Store:**

* At the core lies the **central metadata store**, a distributed system responsible for storing all metadata about your data assets. This store ensures consistency and eliminates siloed information across workspaces.

**Object Hierarchy:**

* Three primary levels exist:
  + **Metastore:** The top level, housing multiple catalogs.
  + **Catalog:** Groups data assets based on projects, departments, etc.
  + **Schema (Database):** Contains tables, views, and volumes within a catalog.
  + **Table/View/Volume:** Represent individual data storage units.

**Microservices Architecture:**

* Unity Catalog utilizes a **microservices architecture**, breaking down functionalities into modular services for flexibility and resilience. These services include:
  + **Catalog Service:** Manages catalogs, schemas, and tables.
  + **Lineage Service:** Tracks data transformation history.
  + **Access Control Service:** Enforces permission checks.
  + **Auditing Service:** Logs access and usage data.

**Centralized API:**

* A single, **unified API** provides access to all services, offering clients (notebooks, Spark jobs, SQL queries) a consistent way to interact with the object model.

**Clients and Interactions:**

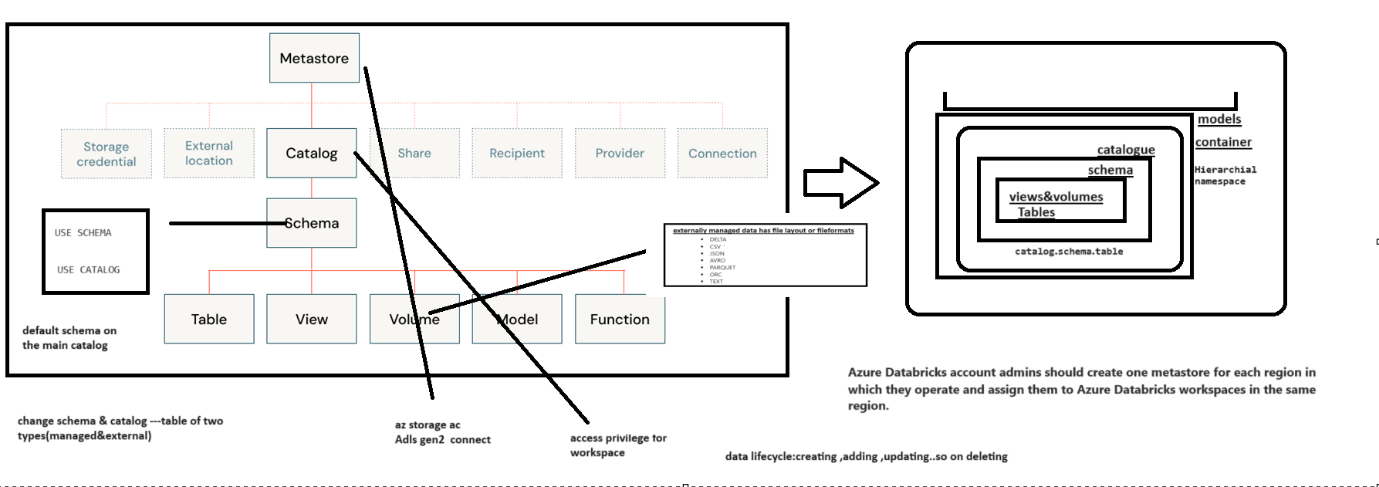
* Clients issue requests through the API.
* The API directs requests to relevant services.
* Services process requests and return results or updates.

**Security and Scalability:**

* Secure communication channels protect data access.
* Distributed infrastructure ensures high availability and scalability for handling large data volumes.

**Key Features within the Architecture:**

* **Open APIs:** Support integration with existing data catalogs, governance tools, and cloud platforms.
* **Connectors:** Pre-built connectors simplify integration with popular data storage systems and frameworks.
* **Lineage Support:** Tracks data transformations for impact analysis and quality management.
* **Fine-grained Access Control:** Grants permissions at various levels (catalogs, schemas, tables) for enhanced data security.



# **unity-catalog-quickstart-python(Python)**

# Create a catalog

To create a catalog, use the CREATE CATALOG command with spark.sql. You must be a metastore admin or user with the CREATE CATALOG privilege on the metastore to create a catalog. If your workspace was enabled for Unity Catalog by default, then workspace admins have the CREATE CATALOG privilege by default.

If your workspace was enabled for Unity Catalog by default, then there may be no managed storage location for the metastore, and you must create a location for the new catalog.

The following commands show how to:

1. Create a catalog.
2. Select a catalog.
3. Show all catalogs.
4. Grant permissions on a catalog.
5. Show all grants on a catalog.

# Create a catalog.

spark.sql("CREATE CATALOG IF NOT EXISTS quickstart\_catalog")

# Create a catalog and specify the managed location

# spark.sql("CREATE CATALOG IF NOT EXISTS quickstart\_catalog MANAGED LOCATION '<location-path>'")

# Set the current catalog.

spark.sql("USE CATALOG quickstart\_catalog")

# Show all catalogs in the metastore.

display(spark.sql("SHOW CATALOGS"))

# Grant create and use catalog permissions for the catalog to all users on the account.

# This also works for other account-level groups and individual users.

spark.sql("""

GRANT CREATE, USE CATALOG

ON CATALOG quickstart\_catalog

TO `account users`""")

# Show grants on the quickstart catalog.

display(spark.sql("SHOW GRANT ON CATALOG quickstart\_catalog"))

# **Create and manage schemas (databases)**

Schemas, also called databases, are the second level of the Unity Catalog three-level namespace. Schemas logically organize tables and views.

# Create a schema in the catalog that was set earlier.

spark.sql("""

CREATE SCHEMA IF NOT EXISTS quickstart\_schema

COMMENT 'A new Unity Catalog schema called quickstart\_schema'""")

# Show schemas in the catalog that was set earlier.

display(spark.sql("SHOW SCHEMAS"))

# Describe the schema.

display(spark.sql("DESCRIBE SCHEMA EXTENDED quickstart\_schema"))

# Grant create table, and use schema permissions for the schema to all users on the account.

# This also works for other account-level groups and individual users.

spark.sql("""

GRANT CREATE TABLE, USE SCHEMA

ON SCHEMA quickstart\_schema

TO `account users`""")

# **Create a managed table**

Managed tables are the default way to create table with Unity Catalog. The table is created in the managed storage location configured for the metastore, catalog, or schema.

The following commands show how to:

1. Select a schema.
2. Create a managed table and insert records into it.
3. Show all tables in a schema.
4. Describe a table.

# Set the current schema.

spark.sql("USE quickstart\_schema")

# Show the current database (also called a schema).

spark.catalog.currentDatabase()

# Create a managed Delta table in the catalog that was set earlier.

spark.sql("CREATE OR REPLACE TABLE quickstart\_table (id STRING)")

# Grant select and modify permissions for the table to all users on the account.

# This also works for other account-level groups and individual users.

spark.sql("""

GRANT SELECT, MODIFY

ON TABLE quickstart\_table

TO `account users`""")

# List the available tables in the catalog that was set earlier.

display(spark.sql("SHOW TABLES"))

# Insert 10 rows into the table.

spark.range(10).selectExpr("id").write.insertInto("quickstart\_table")

# Show the table.

display(spark.table("quickstart\_table"))

# Show all of the available tables in the schema.

display(spark.sql("SHOW TABLES in quickstart\_schema"))

**Notes:**

