Databricks Unity Catalog Implementation @ High Level

1. Overview

Databricks Unity Catalog is a unified governance solution for all data assets in your Databricks workspace. It provides:

1.Centralized metadata management:
A single metastore to catalog and organize all data assets across your Databricks environment.

- Data assets across your Databricks environment include tables, views, files, functions, models, and other structured or unstructured datasets registered in the metastore.

2.Fine-grained access controls (table, column, row):
precisely control who can access specific data at any level of detail.

- Specific data at any level of detail refers to entire tables, individual columns, or filtered rows containing sensitive or business-critical information

3.Data lineage and audit logging:
Automatically track data flow and user actions for transparency and compliance.

- Databricks tracks data flow through lineage of queries and transformations, and logs user actions like reads, writes, permission changes, and schema modifications.

4.Consistent security model across workspaces:
Enforce uniform access policies and governance in every workspace.

- Uniform access policies and governance enforce consistent permissions, security rules, and compliance standards across all workspaces and data assets.

2. Step-by-Step Navigation Guide

Below is a clear sequence of steps for configuring and using Unity Catalog:

2.1 Prerequisites

- Databricks Premium or Enterprise plan
- Metastore admin role assigned
- Workspace(s) attached to your account
- Databricks CLI configured

2.2 Create the Unity Catalog Metastore

1. **Open Databricks Admin Console**
    - Go to **Account Console** > **Data**.
2. **Create Metastore**
    - Click **Create Metastore**.
    - Provide:
        - Name
        - Storage root path (e.g., an S3 bucket or ADLS)
        - Region

3. **Configure Managed Storage Credential**
    - Use Databricks-generated IAM role or create your own.

2.3 Assign Workspaces to the Metastore

1. From the Metastore settings, click **Assign to Workspace**.
2. Select the workspace(s) to attach.
3. Confirm the assignment.

2.4 Configure Access Control

1. In **Data** tab, navigate to **Grants**.
2. Assign **Data Steward** and **Data Owner** roles.
3. Use SQL Grants:

```
sql
CopyEdit
GRANT USAGE ON CATALOG <catalog_name> TO `group_name`;
GRANT SELECT ON SCHEMA <schema_name> TO `group_name`;
```

2.5 Create Catalogs, Schemas, and Tables

1. **Create Catalog:**

```
sql
CopyEdit
CREATE CATALOG sales_data;
```

2. **Create Schema:**

```
sql
CopyEdit
CREATE SCHEMA sales_data.monthly_reports;
```

3. **Create Table:**

```
sql
CopyEdit
CREATE TABLE sales_data.monthly_reports.june (
  order_id STRING,
  amount DOUBLE
);
```

2.6 Enable Data Lineage

- Unity Catalog automatically captures lineage.
- Access via the **Data Explorer > Lineage** tab.

2.7 Configure Row and Column-Level Security

1. **Row Filter Example:**

```
sql
CopyEdit
CREATE OR REPLACE ROW FILTER filter_region
AS (region = 'US');
sql
CopyEdit
ALTER TABLE sales_data.monthly_reports.june
```

SET ROW FILTER filter_region;

2. **Column Mask Example:**

sql
CopyEdit
CREATE OR REPLACE FUNCTION mask_ssn(ssn STRING)
RETURNS STRING
RETURN CONCAT('XXX-XX-', RIGHT(ssn, 4));
sql
CopyEdit
ALTER TABLE customer_data
ALTER COLUMN ssn
SET MASK mask_ssn;

2.8 Monitor Audit Logs

- Enable audit logs in your workspace.
- Integrate with cloud-native logging (AWS CloudTrail, Azure Monitor).

3. Best Practices by Project Complexity

Simple Projects

(e.g., single catalog, small team)

- Use one catalog to group all data.
- Start with **table-level permissions**.
- Keep schema naming consistent.
- Avoid over-segmentation.
- Document grants in a shared spreadsheet.

Medium Projects

(e.g., multiple teams, multiple schemas)

- Create separate catalogs per domain or business unit.
- Use **schemas** to isolate environments (dev, test, prod).
- Implement **column masking** for sensitive fields.
- Leverage **groups** rather than individual users for grants.
- Enforce naming conventions (<team>_<purpose>_<environment>).
- Automate grants with Terraform or Databricks CLI.

Complex Projects

(e.g., enterprise data mesh, multi-region)

- Adopt a **catalog per domain model** (data mesh architecture).
- Implement **row-level security policies** for multi-tenant data.
- Enable **data lineage** and integrate with external catalogs (Purview, Glue).
- Standardize all permissions in Infrastructure-as-Code.
- Use CI/CD pipelines for schema/table creation.
- Periodically review and prune obsolete grants.
- Consider **schema evolution policies** for big data ingestion.

4. Reference Architecture Diagram (Textual)
[Account Level]

Yaml

```
[Unity Metastore]
   ├── Catalog: finance
   │      ├── Schema: dev
   │      ├── Schema: prod
   │      └── Schema: test
   ├── Catalog: sales
   │      └── Schema: prod
   └── Catalog: hr
          └── Schema: sensitive_data
```

[Workspace Level]
vbnet
CopyEdit
Workspace A:
Linked to Unity Metastore
Cluster with Unity Catalog enabled

Workspace B:
Linked to Unity Metastore
Cluster with Unity Catalog enabled

5. Security & Governance

- All access governed by the **Metastore Admins**.
- Leverage **SCIM** to sync user groups.
- Rotate credentials periodically.
- Enable **audit log forwarding**.
- Test permissions before production rollout.

6. Operational Considerations

- Monitor performance of catalogs and metadata queries.
- Plan storage in advance to avoid S3/ADLS permission conflicts.
- Version your schemas and track changes.

7. Resources & Links

- Unity Catalog Docs : https://docs.databricks.com/data-governance/unity-catalog/index.html
- Terraform Provider : ttps://registry.terraform.io/providers/databricks/databricks/latest/docs
- Databricks CLI : https://docs.databricks.com/dev-tools/cli/index.html