Enterprise Databricks Metastore Implementation

1. Document Overview

Title:

Enterprise Databricks Metastore Architecture and Implementation

Author:

Sri Sivakumar Ramar

Date:

July 07

Version: Draft

1.0

2. Objective

Define the architectural design, setup, governance, security, and operational considerations for implementing the **Unity Catalog Metastore in Databricks** to support enterprise-grade data governance, lineage, and access control across all workspaces.

3. Scope

- Initial setup of the **Metastore** for all Databricks workspaces
- Integration with cloud storage (e.g., Azure Data Lake Storage Gen2, AWS S3, GCP GCS)
- Fine-grained access controls (table, view, schema)
- Non-functional requirements, including **security**, **availability**, **scalability**, and **compliance**

4. Architectural Overview

4.1 Key Components

Component	Description
Unity Catalog Metastore	Central governance layer for metadata, access policies, data lineage, and auditing.
External Locations	Defined paths in cloud storage where tables are stored (managed or external).
Storage Credentials	Secure authentication to storage accounts/buckets (e.g., Azure Service Principal, AWS IAM Role).
Catalogs/Schemas	Logical containers to organize data assets (Catalog > Schema > Table/View).
Databricks Workspaces	Environments where users consume and manage data, all federated through the same Metastore.
Audit & Lineage	Automatically tracks user actions and data movement across the Metastore.
Identity Federation	Centralized user/group permissions integrated with Azure AD / AWS IAM.

5. Design Considerations

5.1 Namespace Strategy

- **Catalogs** by business domain (e.g., finance, sales, hr)
- **Schemas** by sub-domain or lifecycle (e.g., raw, curated, sandbox)
- **Tables** clearly prefixed for clarity (e.g., stg_, dim_, fact_)

5.2 Storage Design

- **External Locations** configured per catalog with **separate storage accounts/containers/buckets**
- Data **encryption at rest** (AES-256)
- **Soft-delete and versioning** enabled at the storage layer

5.3 Security & Access Control

- **Fine-grained RBAC** via Unity Catalog (grants on catalogs, schemas, tables)
- Integration with **Azure Active Directory** or **AWS IAM**
- **Dynamic data masking** for sensitive columns
- **Row-level security policies** as applicable

5.4 High Availability & Disaster Recovery

- Unity Catalog is **region-specific**; design regional strategies if needed
- **Storage replication** and **cross-region disaster recovery**
- Backups of critical metadata exported regularly via Databricks CLI/API

5.5 Cost Management

- Separation of environments (dev/test/prod) with **different storage locations**
- Monitoring **storage consumption** and **access costs**
- Automation of stale data archiving

6. Non-Functional Requirements

NFR Area	Description
Security	- Encryption at rest and in transit - Least privilege model - RBAC enforcement - Audit trails retained minimum 1 year
Scalability	- Support for 1,000+ tables across multiple business units - Elastic storage growth - Metadata performance monitoring
Availability	- SLA target 99.9% uptime - HA cloud storage - Redundant region readiness
Compliance	- GDPR and SOC2 compliance - Data residency adherence - Regular compliance reviews
Observability	- Automated audit logging - Lineage visualization in Unity Catalog UI - Custom monitoring dashboards (e.g., via Databricks SQL or external tools)
Performance	- Metadata query latency <500 ms - Permission evaluation within acceptable thresholds
Cost	- Budget allocation per business unit - Cost attribution tagging - Alerts on thresholds

7. Implementation Steps

7.1 Prerequisites

- Cloud storage provisioned and secured
- Databricks workspace(s) deployed
- Identity provider (Azure AD / AWS IAM) integrated

7.2 Metastore Setup

1. **Create Metastore**
    - Admin runs:

sql

CopyEdit

```
CREATE METASTORE my_enterprise_metastore
```

2. **Assign Metastore**
    - Attach to all workspaces:

sql

CopyEdit

```
ALTER METASTORE my_enterprise_metastore OWNER TO `account_admins`
```

3. **Configure Storage Credential**
    - Azure:

sql

CopyEdit

```
CREATE STORAGE CREDENTIAL my_credential
WITH AZURE_SERVICE_PRINCIPAL (
  CLIENT_ID = 'xxx',
  CLIENT_SECRET = '***',
  DIRECTORY_ID = 'yyy'
)
```

    - AWS:

sql

CopyEdit

```
CREATE STORAGE CREDENTIAL my_credential
WITH IAM_ROLE 'arn:aws:iam::123456789012:role/my-role'
```

4. **Create External Location**

sql

CopyEdit

```
CREATE EXTERNAL LOCATION finance_data
URL 'abfss://finance@mydatalake.dfs.core.windows.net/'
WITH STORAGE CREDENTIAL my_credential
```

5. **Create Catalogs**

sql

CopyEdit

```
CREATE CATALOG finance
MANAGED LOCATION 'abfss://finance@mydatalake.dfs.core.windows.net/'
```

6. **Grant Access**

sql
CopyEdit
GRANT USAGE ON CATALOG finance TO `finance_users`
GRANT SELECT ON SCHEMA finance.raw TO `data_scientists`

7.3 Governance

- Define **naming conventions**
- Setup **approval workflow** for new tables/schemas
- Configure **data lineage policies**

7.4 Testing & Validation

- Validate permissions inheritance
- Validate data encryption
- Simulate failover scenarios
- Audit query performance

7.5 Operational Handover

- Document playbooks for:
- Catalog/schema/table creation
- Access management
- Monitoring and alerts
- Backup and recovery

8. Risks & Mitigations

Risk	Mitigation
Misconfigured access controls	Enforce peer-review process before applying grants
Single-region Metastore limitations	Plan region-specific metastores or region-failover strategies
Unexpected storage costs	Implement cost monitoring alerts and scheduled reviews
Metadata corruption	Schedule regular exports and version control
Performance degradation under large scale	Periodic metadata optimization and catalog partitioning

9. Future Enhancements

- Enable **automated schema evolution** tracking
- Integrate with **third-party catalog solutions** if needed
- Enrich **lineage visualization** with BI/ETL tool metadata

10. Appendix

- Links to official Databricks Unity Catalog documentation
- Sample policy templates
- Reference to compliance requirements (e.g., GDPR)

Final Notes from Architect Perspective

This design aims to:

- Establish a **single source of truth for metadata**
- Ensure **enterprise-grade security and governance**
- Enable **scalable, cost-effective data management**
- Lay a foundation for **future multi-region and cross-cloud expansion**