Governance at Scale: Databricks Unity Catalog Technical Reference

Use case:

This Databricks Unity Catalog design establishes a secure, scalable, and region-aware data governance framework for an organization with Sales, Marketing, and eCommerce lines of business, supporting approximately 3,000 users distributed across multiple regions.

Each region will have dedicated metastores to ensure data residency compliance, with logically separated catalogs (sales_data, marketing data, ecommerce_data) containing standardized schemas for raw, curated, and analytics data.

Access control is managed centrally through identity federation (e.g., Azure AD) and SCIM-provisioned groups mapped to clearly defined roles with least-privilege permissions.

Infrastructure as Code (Terraform) will automate creation and management of metastores, catalogs, and policies, while centralized audit logging and monitoring ensure compliance and traceability.

This setup provides a consistent, secure foundation for collaborative data management and analytics across all business units.

0. Design Objectives

Why we're doing this:

- Establish a unified data governance layer across all business domains.
- Simplify secure data sharing between Sales, Marketing, and eCommerce.
- Ensure region-specific compliance (e.g., GDPR).
- Centralize lineage, auditing, and discoverability.
- Scale to 3,000+ users with role-based access.

1. Governance Model and Guiding Principles

Separation of Concerns

- Each line of business (LoB) has logically isolated catalogs.
- Shared data lives in dedicated "Cross-LoB" catalogs.

Least Privilege Access

- Users and service principals only access what they require.

Central Control

- Unity Catalog Metastore acts as the single source of truth.

Multi-Region Strategy

- Dedicated metastores per region for compliance.
- Common governance framework across metastores.

Automated Entitlements

- All permissions managed declaratively (Infrastructure as Code).

2. Metastore Architecture

Best Practice: A Unity Catalog *Metastore* is region-specific.

Proposed Setup:

- **Metastore Naming Convention:**
  - metastore_sales_<region>
  - metastore_marketing_<region>
  - metastore_ecommerce_<region>
- **Region Example:**
  - metastore_sales_us_east
  - metastore_sales_eu_west
  - etc.
- **Shared Data:**
  - Optional “Global Shared Metastore” if cross-region replication is needed (advanced).

Diagram (Textual):

```yaml
Regions:
  ├── US East
  │     ├── Metastore: sales_us_east
  │     ├── Metastore: marketing_us_east
  │     └── Metastore: ecommerce_us_east
  └── EU West
        ├── Metastore: sales_eu_west
        ├── Metastore: marketing_eu_west
        └── Metastore: ecommerce_eu_west
```

Best Practice:

Start with one Metastore per region and expand as necessary. Avoid a single Metastore spanning multiple regions due to latency and compliance.

3. Catalog and Schema Design

Logical Catalog Structure per Metastore:

```yaml
Catalog: sales_data
  ├── Schema: raw
  ├── Schema: curated
  └── Schema: analytics

Catalog: marketing_data
  ├── Schema: raw
  ├── Schema: curated
  └── Schema: analytics

Catalog: ecommerce_data
  ├── Schema: raw
  ├── Schema: curated
  └── Schema: analytics
```

Special Catalogs:

- shared_data: Cross-LoB data accessible to all business units.
- sandbox: User experimentation (ephemeral).

Comments:

- **Raw:** Direct ingestion (e.g., Delta Live Tables).
- **Curated:** Cleaned, validated data.
- **Analytics:** BI-ready datasets.

Best Practice:

Use clear naming conventions (<lineofbusiness>_data) and avoid ambiguous names like default.

4. Access Control Design

Approach:

- Use **Unity Catalog GRANT statements** to define permissions.
- Manage access through **Groups**, mapped from your Identity Provider (Azure AD/Okta).

Groups Example:

python-repl

CopyEdit

sales_readers

sales_writers

sales_admins

marketing_readers

...

Role Mapping:

Role	Permissions
Readers	SELECT
Writers	SELECT, INSERT, UPDATE
Admins	ALL PRIVILEGES + OWNERSHIP
Data Stewards	ALL on curated + analytics schemas
Data Engineers	ALL on raw + curated schemas

Sample GRANT statements:

sql

CopyEdit

GRANT USAGE ON CATALOG sales_data TO `sales_readers`;

GRANT SELECT ON SCHEMA sales_data.analytics TO `sales_readers`;

GRANT ALL PRIVILEGES ON SCHEMA sales_data.curated TO `sales_writers`;

Best Practice:

Avoid granting permissions to individual users—use groups consistently.

5. Identity Federation & Authentication

Options:

- Azure Active Directory integration (recommended).
- SCIM provisioning for automatic user and group sync.

Setup Steps:

1. Integrate Databricks workspace with AAD tenant.
2. Enable SCIM connector.
3. Map AAD groups to Databricks groups.
4. Automate group membership management.

Best Practice:

Use service principals for automation pipelines rather than user tokens.

6. Region Strategy and Data Residency

Requirements:

- Keep EU data within EU regions.
- Keep US data within US regions.
- Minimize cross-region traffic.

Design Notes:

- Each region's Metastore manages its own S3/ADLS storage accounts.
- Region-specific workspaces connect to the matching Metastore.
- Optional: Use Delta Sharing for cross-region sharing without copying.

7. Automation and CI/CD

Infrastructure as Code (IaC):

- **Tools:**
    - Terraform (preferred)
    - Databricks CLI
- **Modules to create:**
    - Metastore resources
    - Catalogs and schemas
    - Access control policies

- Storage credentials
- External locations

Example Terraform Module:

hcl

CopyEdit

```
module "sales_metastore" {
  source          = "./modules/unity_catalog_metastore"
  region          = "us-east"
  catalog_names    = ["sales_data"]
  schemas         = ["raw", "curated", "analytics"]
  group_policies   = var.sales_group_policies
}
```

Best Practice:

All grants and resources declared in code—never manually created in the console.

8. Monitoring and Auditing

Tools:

- Unity Catalog audit logs (accessible via the account console).
- Cluster-level audit logs.
- Event Hub / Event Grid (if on Azure).
- Alerts for:
  - Unauthorized access attempts.
  - High-volume exports.

Best Practice:

Centralize audit logs in a dedicated logging workspace or SIEM.

9. Deployment Plan

Identity & Access:

- Integrate AAD/Okta.
- Sync groups via SCIM.

Storage:

- Create region-specific storage accounts/buckets.
- Define storage credentials.

Metastore:

- Deploy Metastore per region.
- Assign to target workspaces.

Catalogs and Schemas:

- Create catalogs (sales_data, marketing_data, etc.).
- Create schemas (raw, curated, analytics).

Access Policies:

- Grant privileges to groups.

Automation:

- Write and test Terraform modules.
- Pipeline deployment.

Testing:

- Validate permissions.
- Validate data residency.

Training:

- Educate users on naming conventions and access.

Go-Live:

- Enable Unity Catalog in production workspaces.
- Monitor initial usage.

This Databricks Unity Catalog design is recommended for:

Ideal Audience For**:**

- **Large or mid-sized enterprises** with **multiple lines of business** (e.g., Sales, Marketing, eCommerce).
- Organizations with **hundreds to thousands of data consumers and producers** (you mentioned ~3,000 users).
- Companies operating in **multiple regions** where:
    - Data residency regulations (GDPR, HIPAA, etc.) apply.
    - Performance and latency need to be optimized locally.

- Enterprises adopting **Lakehouse architecture** that want:
    - Consistent governance across all data assets.
    - Centralized lineage, auditability, and access controls.
    - Separation of duties between data producers and consumers.
- Teams with **dedicated platform engineering, security, and compliance functions** that can maintain:
    - Infrastructure-as-Code (Terraform) deployment pipelines.
    - Managed identity integrations (AAD or Okta).
    - Strict role-based access control policies.
- Organizations looking to **migrate from legacy data lakes or Hadoop clusters to Databricks** with Unity Catalog as the unified governance layer.

Typical Stakeholders Who Should Be Involved:

- **Data Platform Engineering Teams** – to design and deploy the metastore, catalogs, and automation pipelines.
- **Cloud Infrastructure and Security Teams** – to integrate identity providers and storage accounts.
- **Data Governance and Compliance Teams** – to define access policies and ensure regulatory compliance.
- **Business Data Owners** – from Sales, Marketing, and eCommerce to validate catalog structure and naming conventions.

Final Note:

This design is recommended for **enterprises with multi-region operations and large, cross-functional data teams** seeking secure, scalable, and auditable data governance through Databricks Unity Catalog