Unity Catalog Upgrade

Security Configuration

OBJECTIVES

The purpose of today's discussion is to go over best practices around Databricks Security and plan the following:

- Administrative Groups
- SSO and SCIM Integration with the your Identity Provider (e.g. Azure AD, Okta)
- Users, Service Principals, User Groups
- Identity Federation (between Accounts and Workspaces)
- Object Ownership
- Tagging
- Plan to migrate any legacy approaches to Security (Table Access Control, Passthrough to Storage)
- Plan to migrate all workspace level security to Account level (REFACTOR)

Record the decisions made and incorporate into diagrams and design documents for the project deliverables

Administrative Groups

- Recommend Multiple Admin Personas
- Distribute the responsibility
- Flexibility to Centralize or Decentralize Administration

Roles

- **Cloud Administrator**
- Identity Administrator
- Account Administrator
- Metastore Admin
- Data Owner
- Workspace Administrator

Administers underlying cloud resources

- Storage accounts/buckets
- IAM role/service principals/Managed Identities

Roles

- Cloud Administrator
- **Identity Administrator**
- Account Administrator
- Metastore Administrator
- Data Owner
- Workspace Administrator

Administers underlying identity provider service (when in use)

- Identity provider provisions users and groups into the account
- Avoids need to manually create and manage identities

Roles

- Cloud Administrator
- Identity Administrator
- **Account Administrator**
- Metastore Administrator
- Data Owner
- Workspace Administrator

Administers the account

- Creates, deletes, and assigns metastores to workspaces
- Creates, deletes, and assigns users and groups to workspaces
- Integrates account with an identity provider
- Full access to all data objects

Roles

- Cloud Administrator
- Identity Administrator
- Account Administrator
- **Metastore Administrator**
- Data Owner
- Workspace Administrator

Administers a metastore

- Creates and drops catalogs and other data objects
- Grants privileges on data objects
- Changes ownerships of data objects
- Designated by an account administrator

Roles

- Cloud Administrator
- Identity Administrator
- Account Administrator
- Metastore Administrator
- **Data Owner**
- Workspace Administrator

Owns data objects they created

- Creates nested objects
- Grants privileges to others on owned objects
- Changes ownership of owned objects

Roles

- Cloud Administrator
- Identity Administrator
- Account Administrator
- Metastore Admin
- Data Owner
- **Workspace Administrator**

Administers a workspace

- Manages permissions on workspace assets
- Restricts access to cluster creation
- Adds or removes users
- Elevates users permissions
- Grant privileges to others
- Change job ownership

Persona	Databricks In-built Role	Custom Group Recommended	Note
Account Admin (AWS, Azure)	Y	Y	This role is the highest possible level in the Databricks Privilege Hierarchy
Metastore Admin (AWS, Azure)	Y	Y	This role is analogous to a central Data Steward group at the Organization Level
Catalog Admin (AWS, Azure)	N	Y	This role is analogous to the Data Owner for a Business Unit / Environment. We recommend that you create multiple account-level catalog_admin groups, per BU as an example, for fine-grained access control and then transfer ownership of catalogs to respective owner groups, or give the CREATE CATALOG privilege so they can create+own it.
Schema Admin (AWS, Azure)	N	Y	This role is analogous to the Data Owner for a Team within a BU. We recommend that you create multiple account-level schema_admin groups, for fine-grained access control and then transfer ownership of schemas to respective owner groups, or give the CREATE SCHEMA privilege so they can create+own it.
Workspace Admin (AWS, Azure)	Y	Y	This role is the highest possible level in a workspace. We recommend that you create multiple account-level workspace_admin groups, per BU as an example, for more fine-grained control over access and entitlements, and then assign those groups to a workspace with the Workspace Admin role.
Compute Admin (AWS, Azure)	N	Y	Allows the designation of a select group of users who can spin up compute and create pools, while not bound by cluster policy, without needing to give them workspace admin rights. Give this group the 4 entitlements listed on the linked page. We recommend that you create multiple account-level compute_admin groups, per BU as an example, and assign those groups to a workspace with the User role.

Responsibility	Account Admin	Metastore Admin	Catalog Admin	Schema Admin	Workspace Admin	Compute Admin
Create Metastore	Y					
Manage Principals	Y					
Setup SSO	Y					
Create Catalog		Y	Y			
Create Storage Credential	Y					
Create External Location		Y				
Delegate Access to Data		Y	Y	Y		
Manage Compute Access					Y	Y
Assign Principals to Workspace					Y	

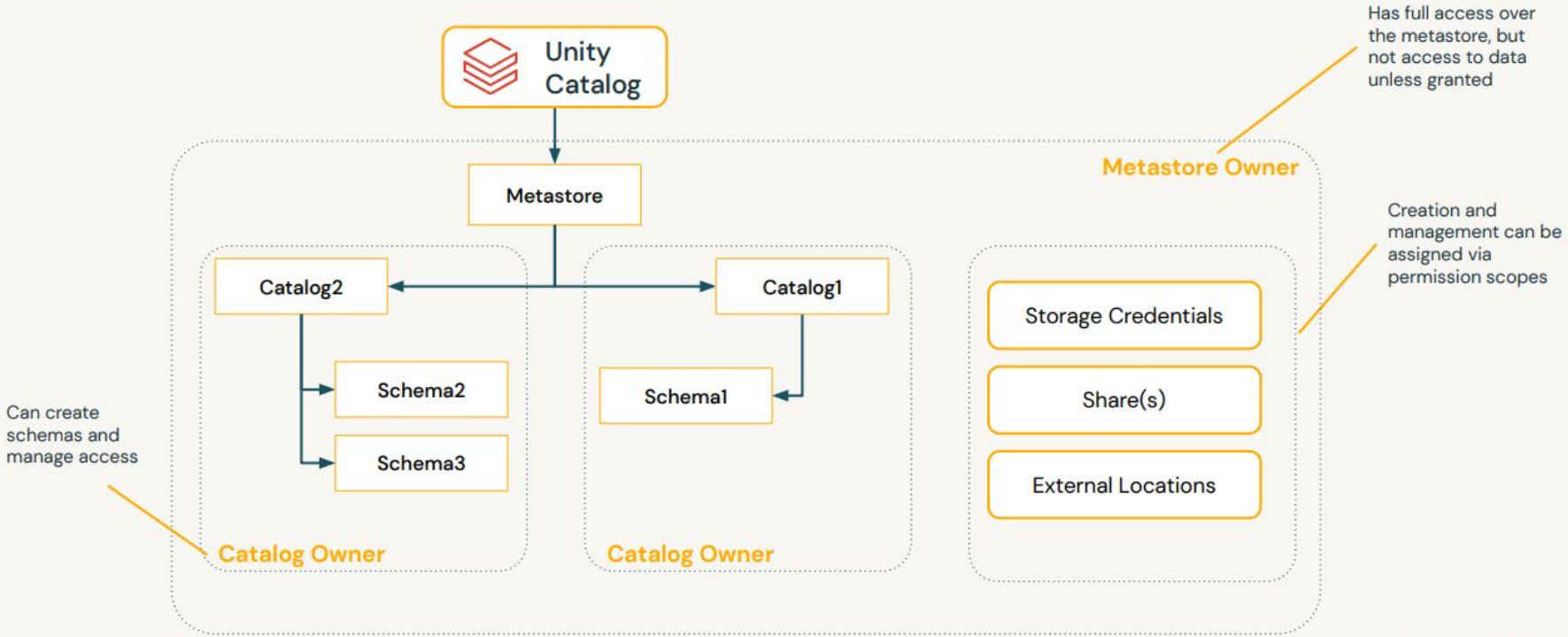
Can do
Should do

Task	Persona	Detail	Status
Collaborate with Identity Admin; Idntify Admin Personas			
Setup SCIM from IDP	Account Admin (+ Identity Admin)	AWS, Azure	☐
Setup SSO		AWS, Azure	☐
Identify Core Admin Personas (Account, Metastore, Workspace)		AWS, Azure	☐
Identify Recommended Admin Personas (Catalog, Compute, Schema)		Admin Personas Tab	☐
Collaborate with Cloud Admin; Create Cloud Resources			
Create Root Bucket	Account Admin (+ Cloud Admin)	AWS, Azure	☐
Create IAM Role (AWS) Create Access Connector Id (Azure)		AWS, Azure	☐
Create a Metastore			
Create Metastore	Account Admin	AWS, Azure	☐
Transfer ownership of metastore to the metastore_admin group		AWS, Azure	☐
Create Storage Credentials & External Locations (for external tables)			
Create Storage Credentials	Account Admin	AWS, Azure	☐
Create External Locations	Metastore Admin	AWS, Azure	☐
Make the workspace UC Enabled			
Create Workspace (if not exists)	Account Admin (AWS) Cloud Admin (Azure)	AWS, Azure	☐
Metastore Assignment (assign metastore to the WS)	Account Admin	AWS, Azure	☐
Workspace Permission Assignment (assign principals to the WS)	Workspace Admin	AWS, Azure	☐

Create Catalog			
Create Catalog	Metastore Admin (or Catalog Admin)	AWS, Azure	☐
Assign Privileges to Catalog			
Transfer Ownership	Metastore Admin	AWS, Azure	☐
Create Securables	Catalog Admin	AWS, Azure	☐
Assign Share Privileges to Catalog securables (optional)			
Create Share	Metastore Admin	AWS, Azure	☐
Assign Share Privileges	Metastore Admin	AWS, Azure	☐

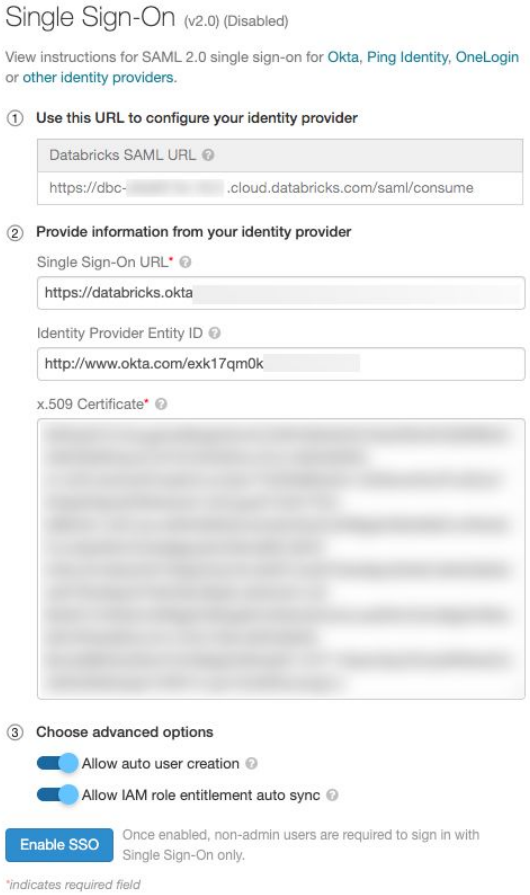
Flexible Organizational Governance

Use centralized or distributed governance approaches



SSO & SCIM Provisioning

Single Sign On (SSO)



What is SSO in Databricks?

- Allows authentication via your organization's identity provider (IdP).
- Supports SAML 2.0 and OpenID Connect (OIDC).

Unified Login

- Manages a single SSO configuration for both account and Databricks workspaces.
- Enabled by default for accounts created after June 21, 2023.

How to Configure

Account Admin Access: Log in to the Databricks account console.
Settings: Navigate to the Single Sign-On tab.
Protocol Selection: Choose between OIDC or SAML 2.0.
IdP Configuration: Create a new client application in your IdP and copy necessary credentials.
Databricks Configuration: Input the copied credentials in Databricks.
Testing and Enabling: Test SSO and then enable it.

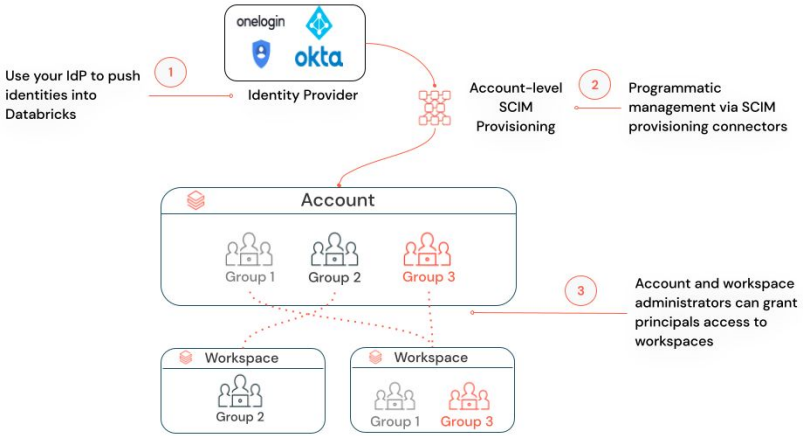
Important Notes

- For accounts created before June 21, 2023, the account owner can still login using username and password.
- REST API calls can still be made using username and password; however, using personal access tokens is recommended.

Risk Mitigation

- Keep the account console open in a different browser during SSO testing to avoid being locked out.

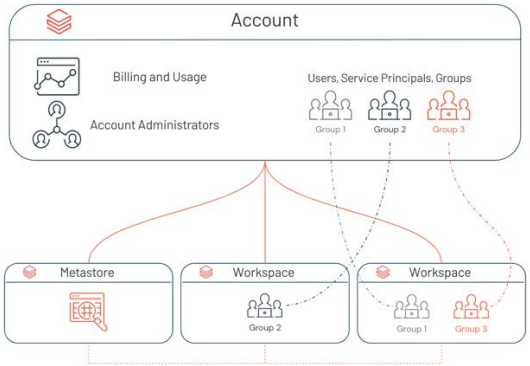
SCIM Provisioning



- Use the System for Cross-domain Identity Management (SCIM) to provision users and groups in Databricks via your identity provider (IdP).
- SCIM: An open standard allowing automation of user provisioning. It enables the creation, update, and removal of users in Databricks based on their status in the organization.
- SCIM Provisioning Levels:
  - Account-level: Recommended by Databricks. Manages all users from the account.
  - Workspace-level: For non-identity federated workspaces or mixed scenarios.
- Requirements:
  - Must have a Databricks Premium plan or higher.
  - Databricks account admin rights for SCIM provisioning to Databricks account.
  - Workspace admin rights for SCIM provisioning to a Databricks workspace.
  - Limits: 10,000 combined users/service principals, 5,000 groups per account/workspace.
- Notes:
  - Using SCIM can override changes made in Databricks settings.
  - SCIM settings in the IdP can override entitlement changes made in Databricks.
- How to Provision:
  - Via IdP: Use a SCIM provisioning connector or the SCIM APIs.
  - Directly: Use the SCIM APIs for account-level or workspace-level provisioning.
- Provision to Databricks Workspace: Workspace-level SCIM is for non-identity federated workspaces. It does not recognize account groups of identity federated workspaces.
- Migration: If moving from workspace-level to account-level provisioning:
  - Create a group in the IdP with all users/groups provisioned using workspace-level SCIM connectors.
  - Set up a new SCIM provisioning connector at the account level.
  - Turn off old workspace-level connectors.
  - Convert workspace-local groups to account groups.

Identity Management

Databricks Identity Management



Identities:

- Users: Identified by email addresses.
- Service Principals: For automated tools, CI/CD platforms. Recommended for Workflows/Jobs
- Groups: For managing access; two types: account groups and workspace-local groups.

Limits:

- Up to 10,000 combined users and service principals.
- Up to 5,000 groups per account/workspace.

Who Can Manage Identities:

- Account Admins (Account Level)
- Workspace Admins (Workspace Level)
- Group Managers, Service Principal Managers

Features & Operations:

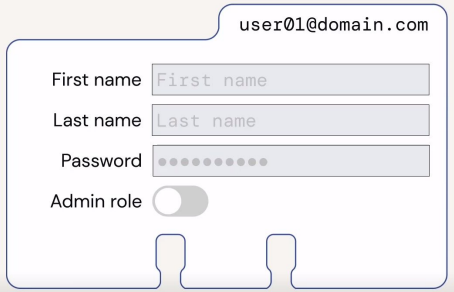
- Identity Federation: Allows syncing between account and workspace.
- Users and Service Principals sync automatically between account and workspaces. Groups created in workspaces (workspace-local groups) do not.
- Entitlements: Define what users can do in Databricks. Done at the workspace level (e.g., SQL access, cluster creation).
- Single Sign-On (SSO): Integrated with third-party identity providers like Okta.

Special Notes:

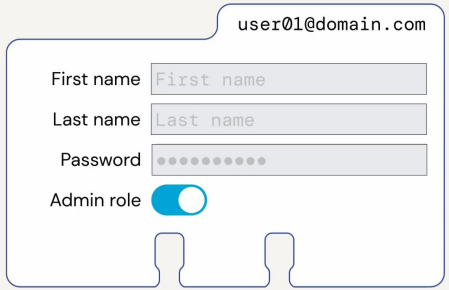
- Unified login (for accounts created after June 21, 2023) allows one SSO config for the account and all workspaces..
- Workspace-local groups are not synced to the account level and require migration.

Identities

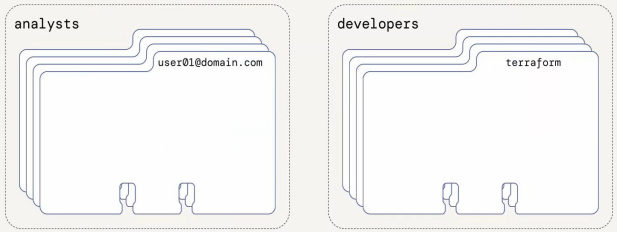
Identities
User



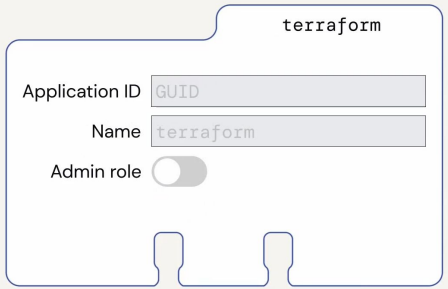
Identities
Account administrator



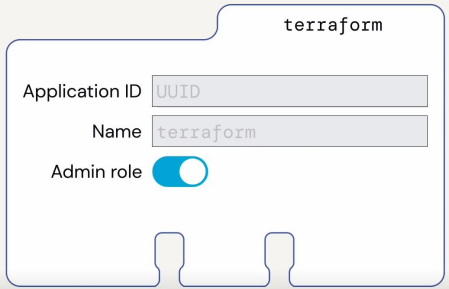
Identities
Groups



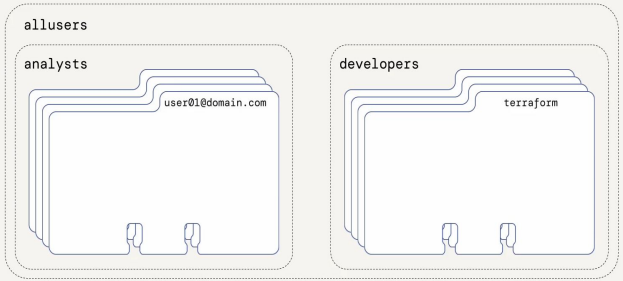
Identities
Service Principal



Identities
Service Principal with administrative privileges



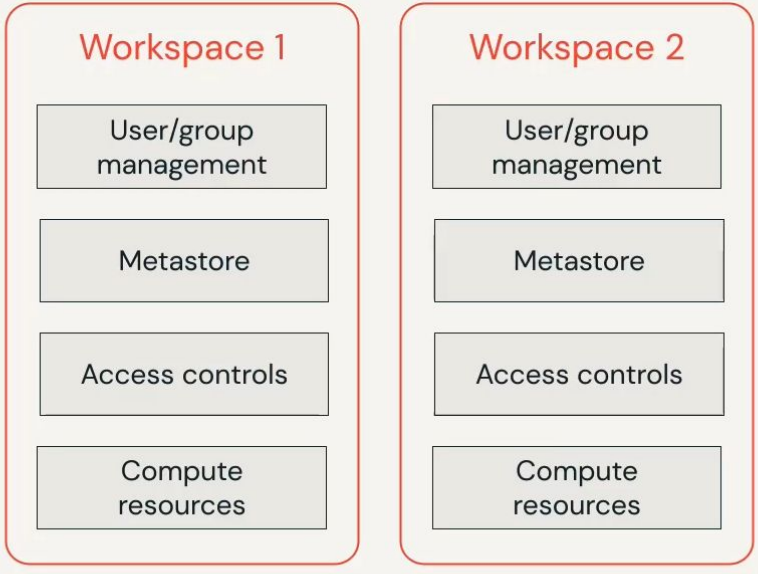
Identities
Nesting groups



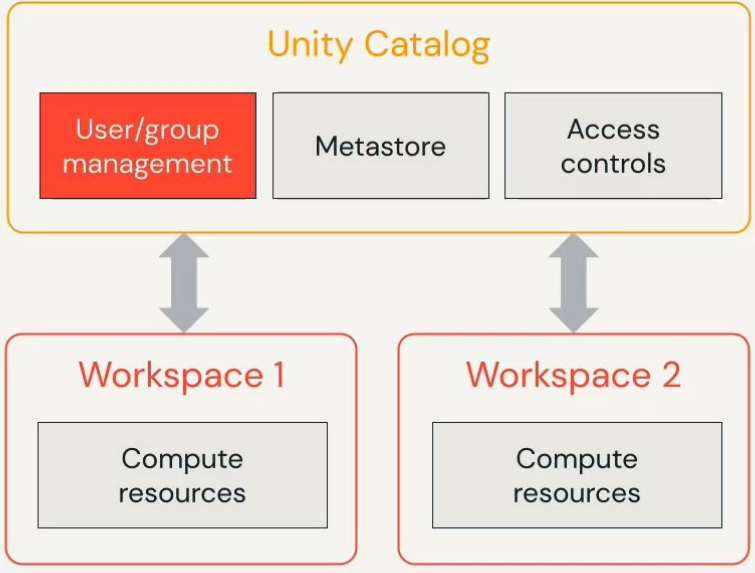
Identities

Recap

Before Unity Catalog

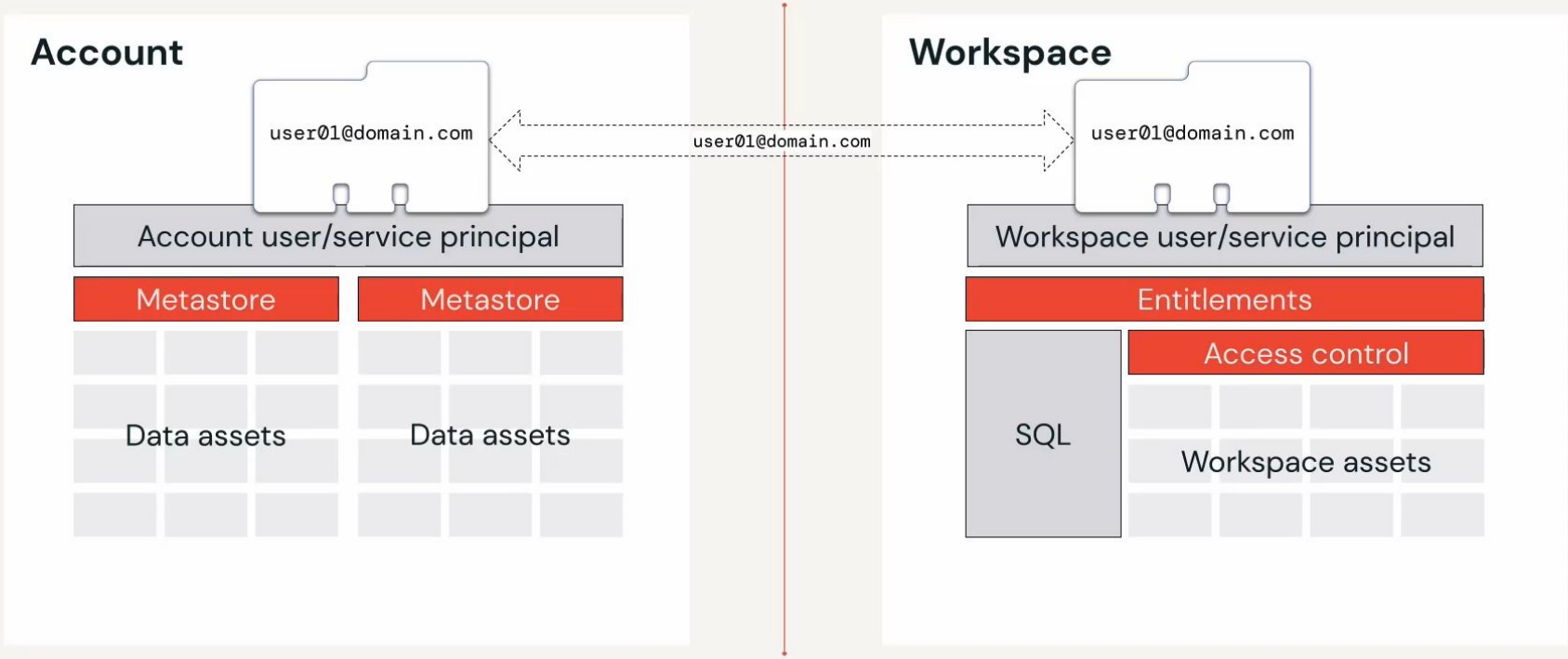


With Unity Catalog



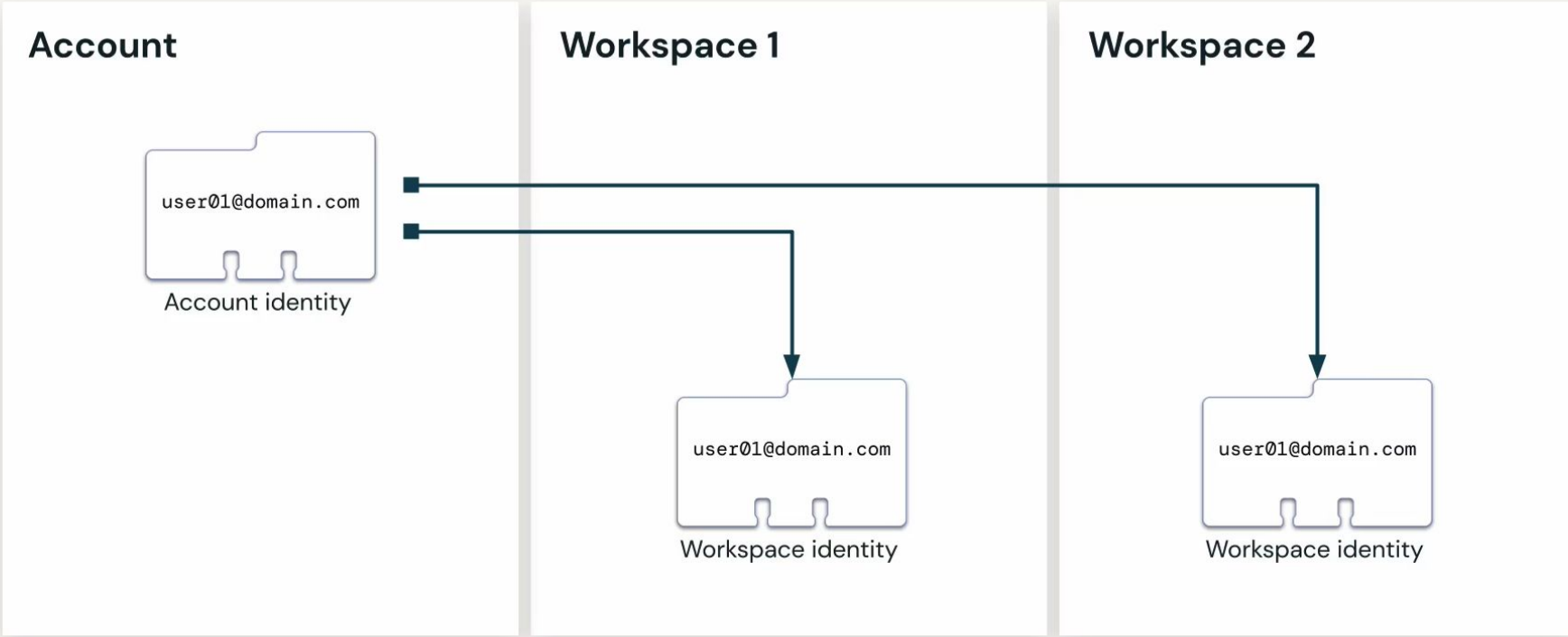
Identities

Relating account and workspace identities



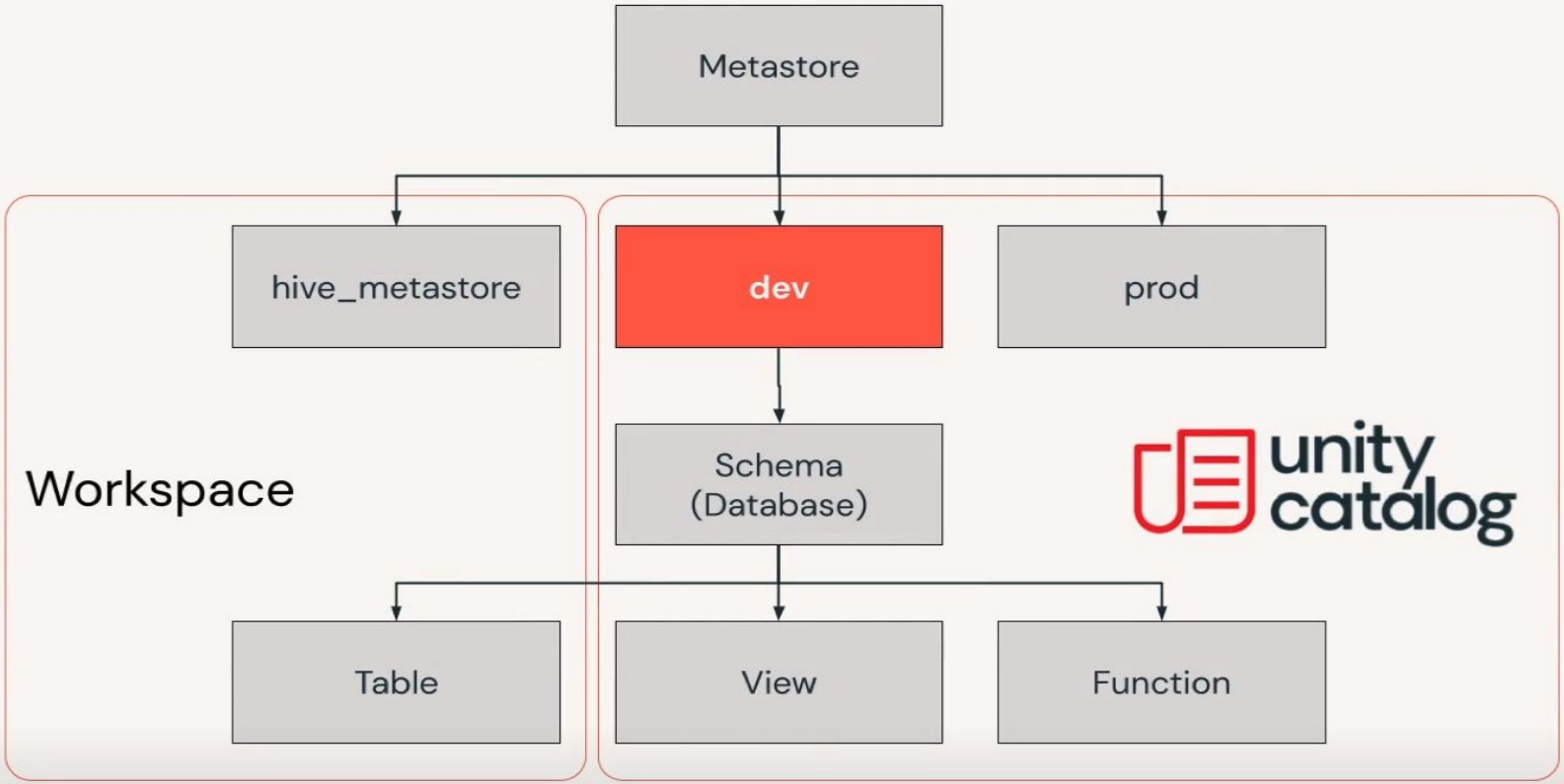
Identities

Identity federation



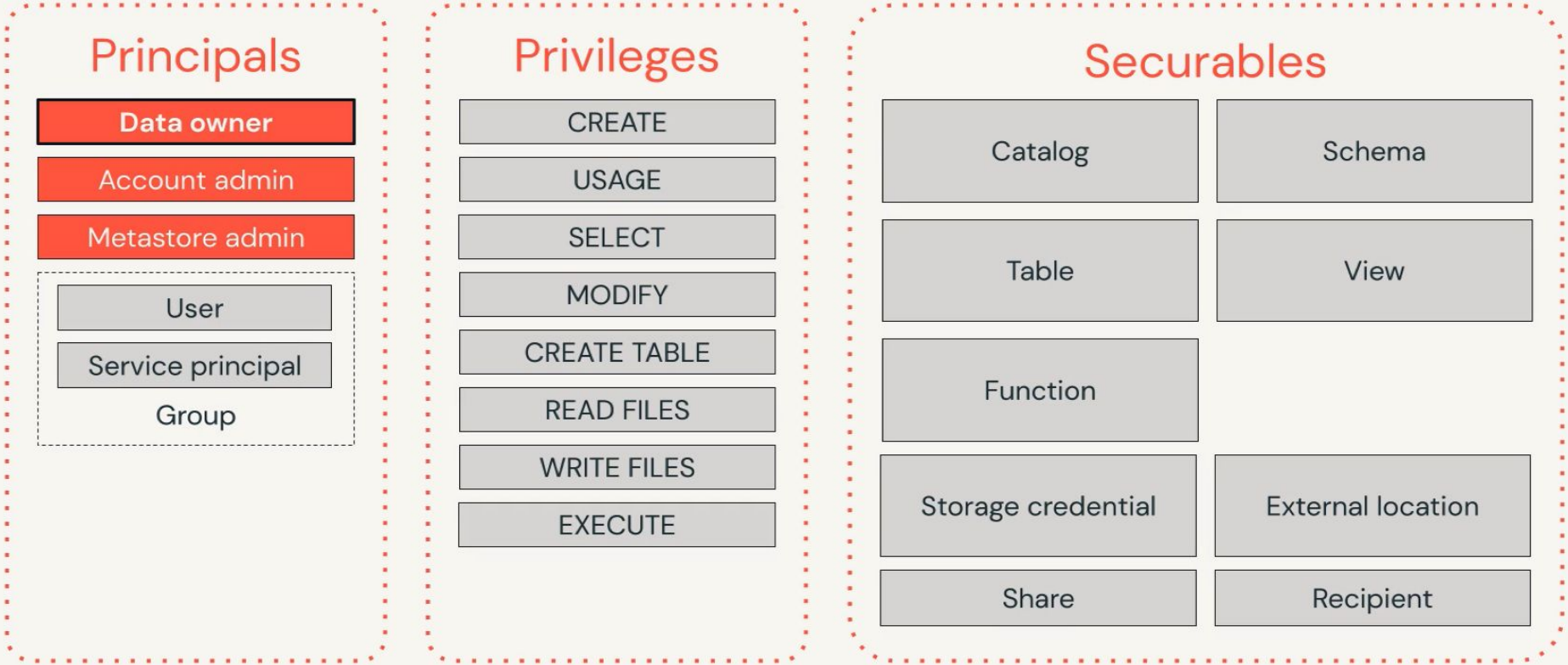
Three-level Namespace

Accessing legacy Hive metastore

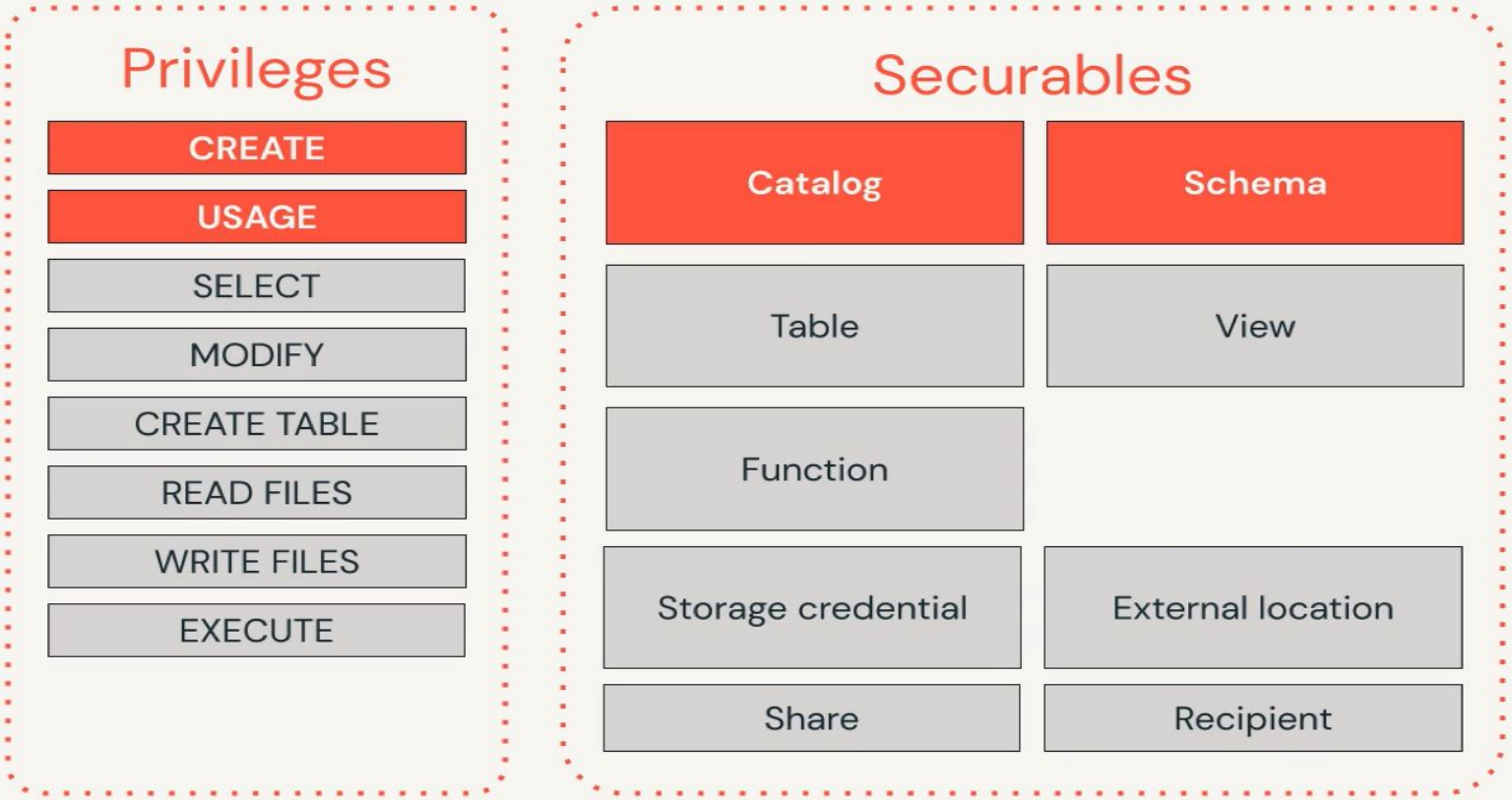


Security Model / Permissions

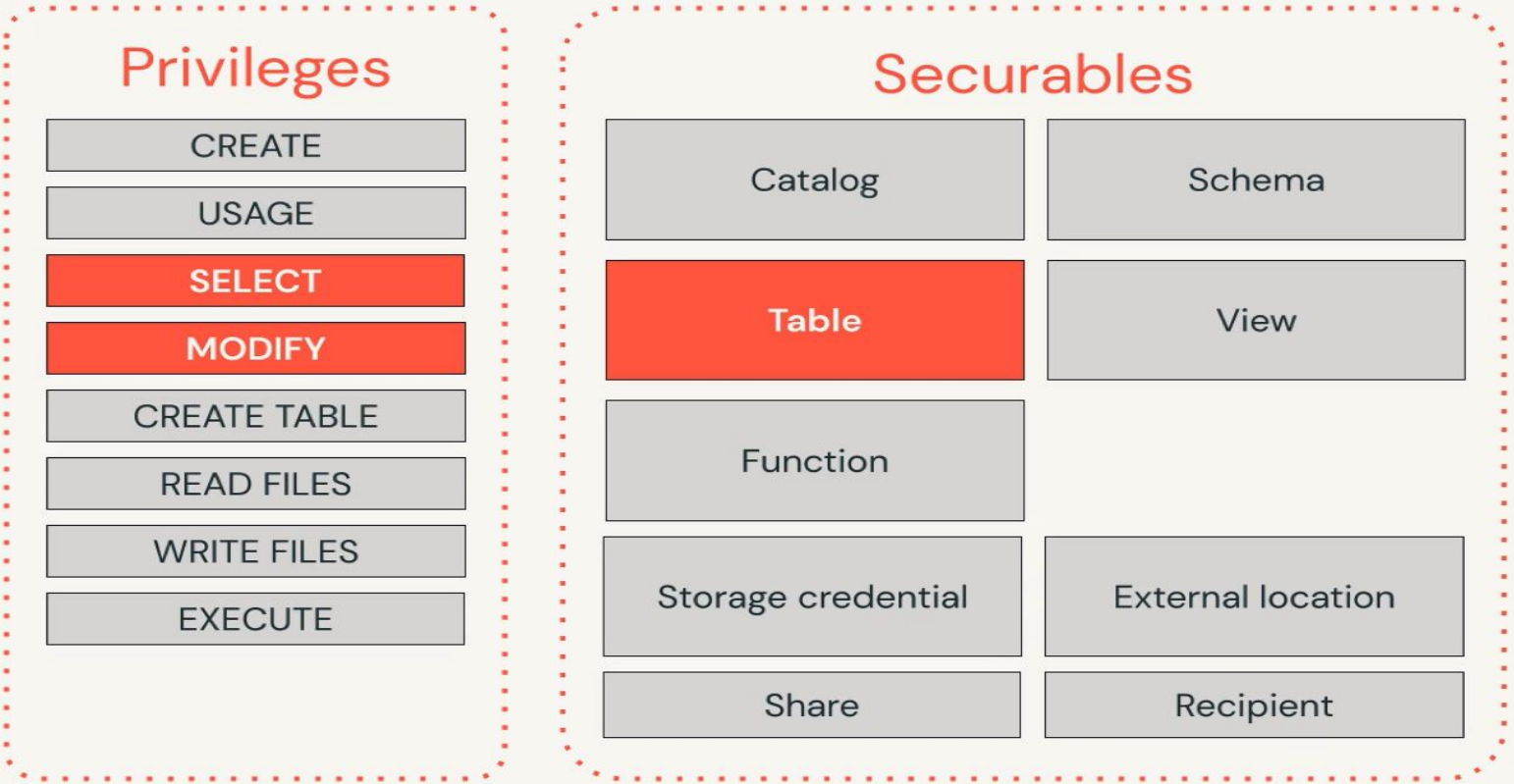
Security model



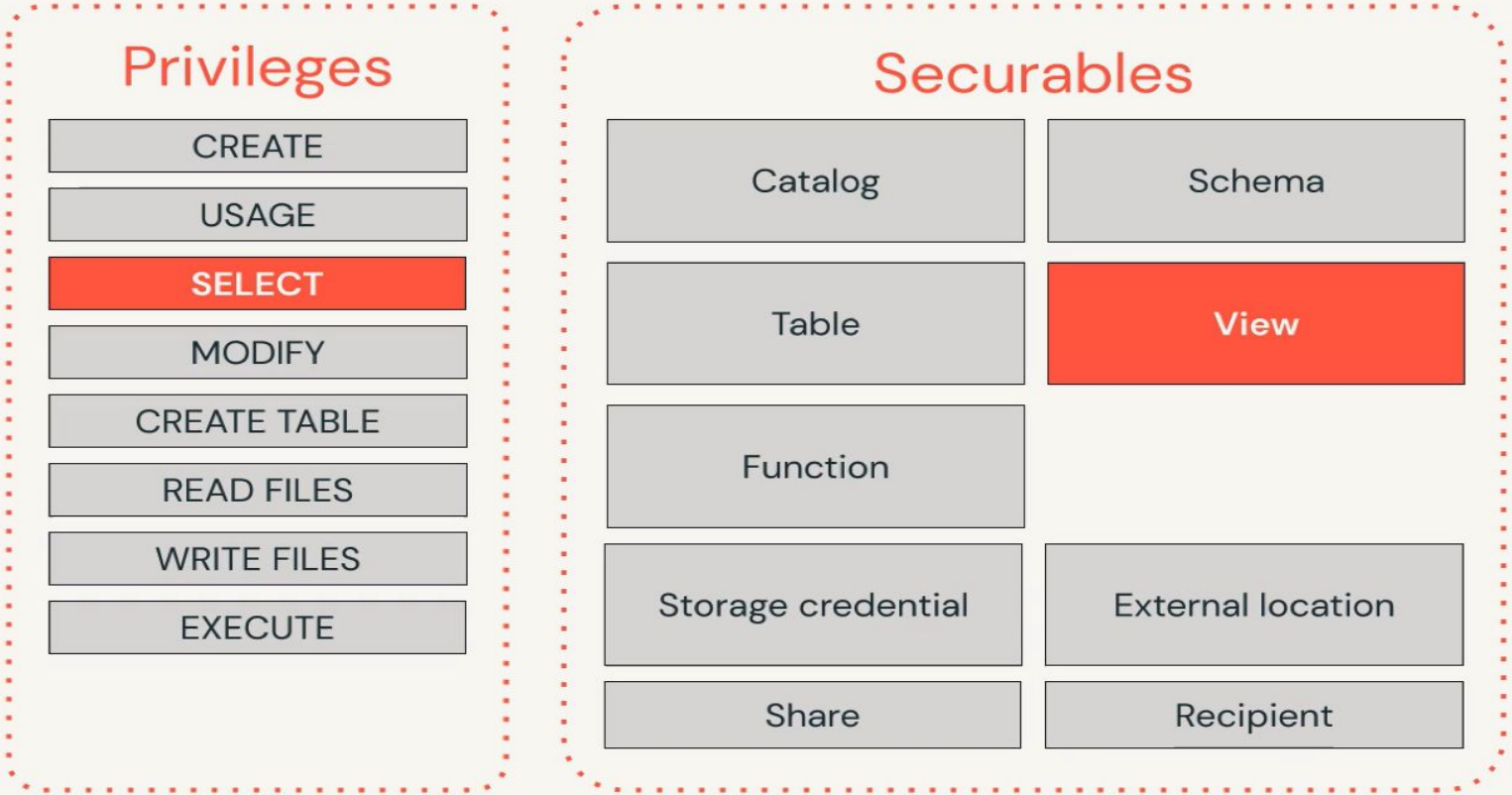
Security model



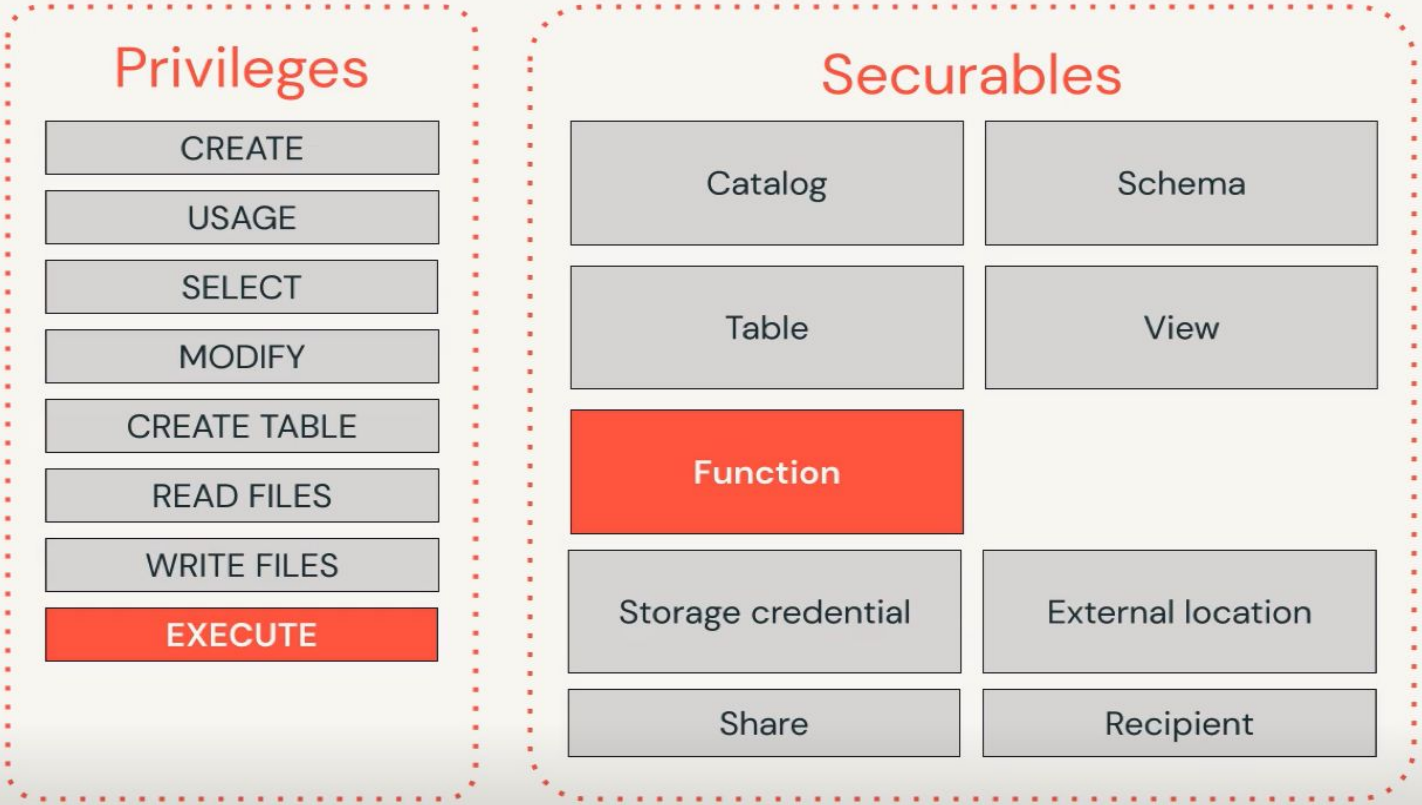
Security model



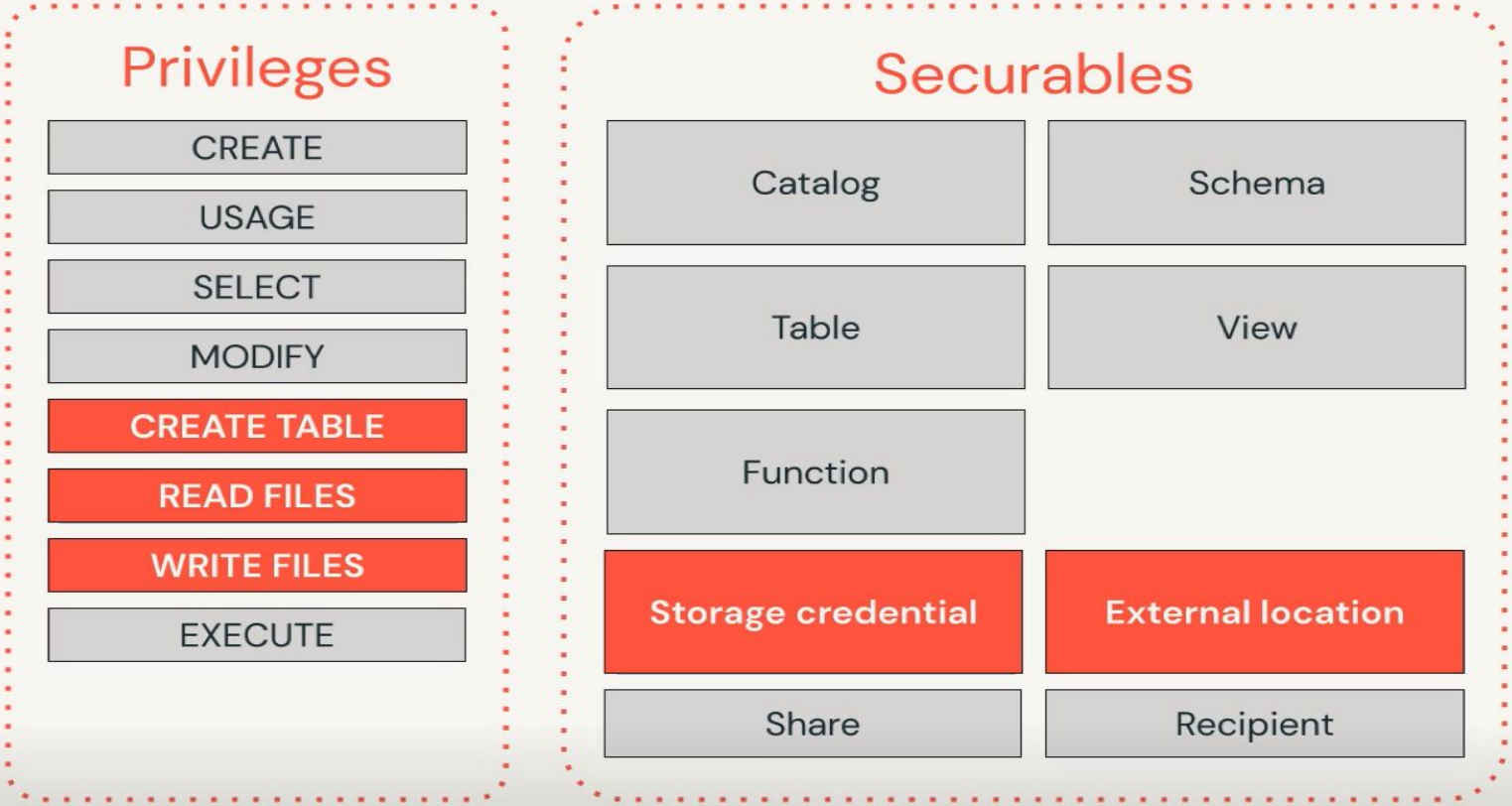
Security model



Security model

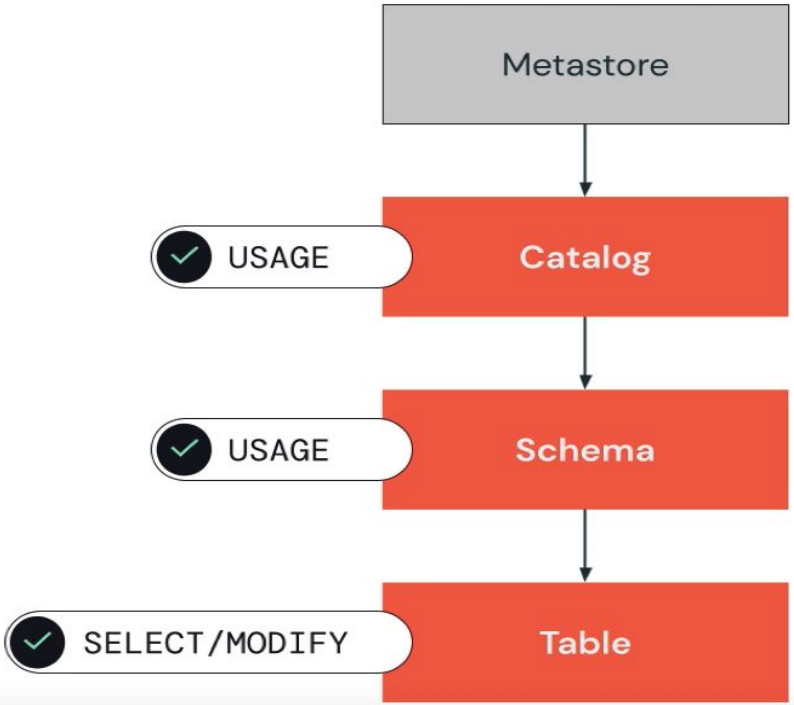


Security model



Privilege Recap

Tables



Querying tables (**SELECT**)

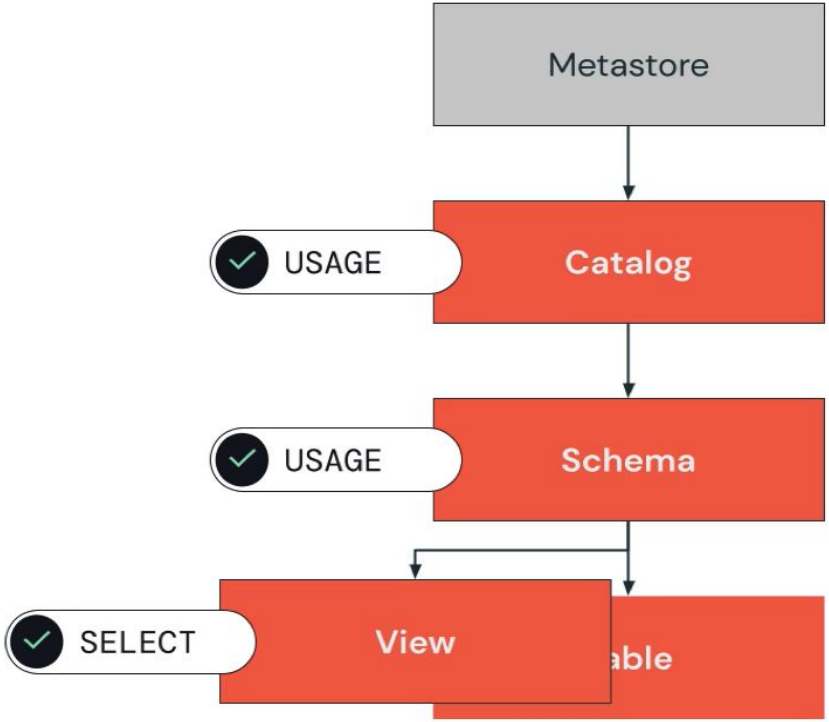
Modifying tables (**MODIFY**)

- Data (INSERT, DELETE)
- Metadata (ALTER)

Traverse container (**USAGE**)

Privilege Recap

Views



Abstract complex queries

- Aggregations
- Transformations
- Joins
- Filters

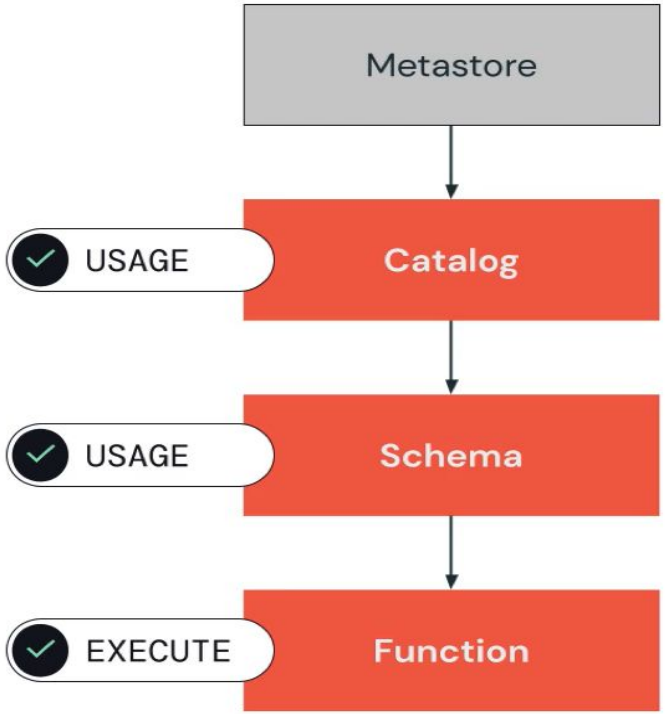
Enhanced table access control

Querying views (**SELECT**)

Traverse container (**USAGE**)

Privilege Recap

Functions



Provide custom code via user-defined functions

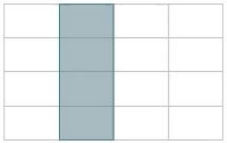
Using functions (**EXECUTE**)

Traverse container (**USAGE**)

Dynamic Views

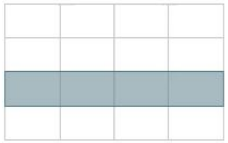
Limit access to columns

Omit column values from output



Limit access to rows

Omit rows from output



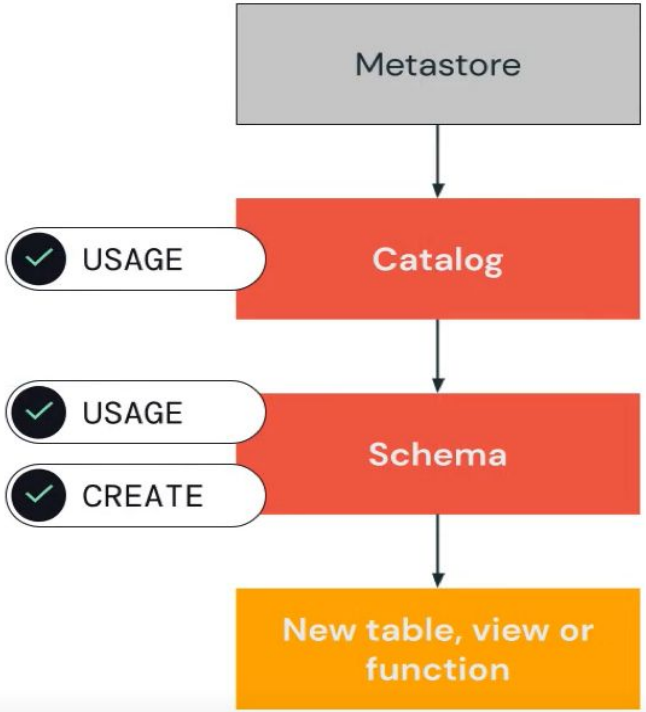
Data Masking

Obscure data

●●●●●●@databricks.com

Can be conditional on a specific user/service principal or group membership through Databricks-provided functions

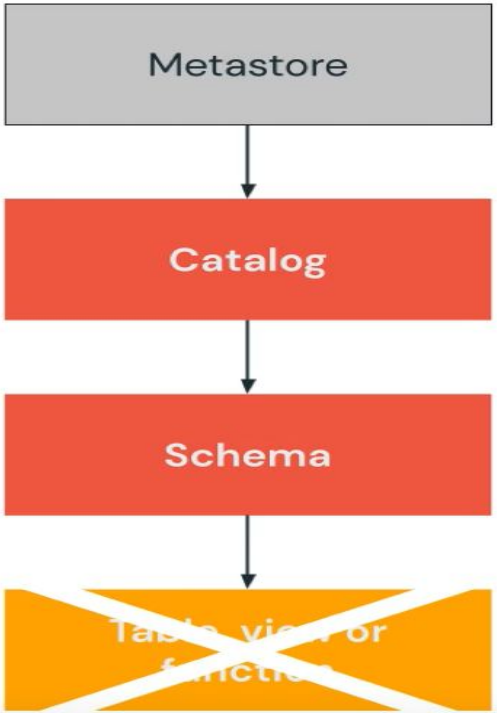
Creating New Objects



Create new objects (**CREATE**)

Traverse container (**USAGE**)

Deleting Objects



DROP objects

Storage Credentials and External Locations

Access Control

CREATE TABLE	READ FILES	WRITE FILES
Create an External Table directly using this Storage Credential	Read files directly using this Storage Credential	Write files directly using this Storage Credential
Storage Credential		
External Location		
Create an External Table from files governed by this External Location	Read files governed by this External Location	Write files governed by this External Location

Access Control

```
GRANT USE CATALOG ON CATALOG < catalog_name > TO < group_name >;
GRANT USE SCHEMA ON SCHEMA < catalog_name >.< schema_name >
TO < group_name >;
GRANT
SELECT
  ON < catalog_name >.< schema_name >.< table_name >;
TO < group_name >;
```

```
CREATE VIEW < catalog_name >.< schema_name >.< view_name > as
SELECT
  id,
  CASE WHEN is_account_group_member(< group_name >) THEN email ELSE 'REDACTED' END AS email,
  country,
  product,
  total
FROM
  < catalog_name >.< schema_name >.< table_name >;
GRANT USE CATALOG ON CATALOG < catalog_name > TO < group_name >;
GRANT USE SCHEMA ON SCHEMA < catalog_name >.< schema_name >.< view_name >;
TO < group_name >;
GRANT
SELECT
  ON < catalog_name >.< schema_name >.< view_name >;
TO < group_name >;
```

```
CREATE VIEW < catalog_name >.< schema_name >.< view_name > as
SELECT
  *
FROM
  < catalog_name >.< schema_name >.< table_name >
WHERE
  CASE WHEN is_account_group_member(managers) THEN TRUE ELSE total <= 1000000 END;
GRANT USE CATALOG ON CATALOG < catalog_name > TO < group_name >;
GRANT USE SCHEMA ON SCHEMA < catalog_name >.< schema_name >.< table_name >;
TO < group_name >;
GRANT
SELECT
  ON < catalog_name >.< schema_name >.< table_name >;
TO < group_name >;
```

Ownership & Privileges:

- Each securable object has an owner with full privileges.
- Best Practice: Assign ownership to the admin group responsible for grant management.

Inheritance:

- Privileges are inherited downward in the object hierarchy. (from catalog, to schema, to table)

Required Privileges for Data Reading:

- SELECT on table/view
- USE SCHEMA on owning schema
- USE CATALOG on owning catalog

Common Scenarios:

- Restrict schema usage to specific teams.

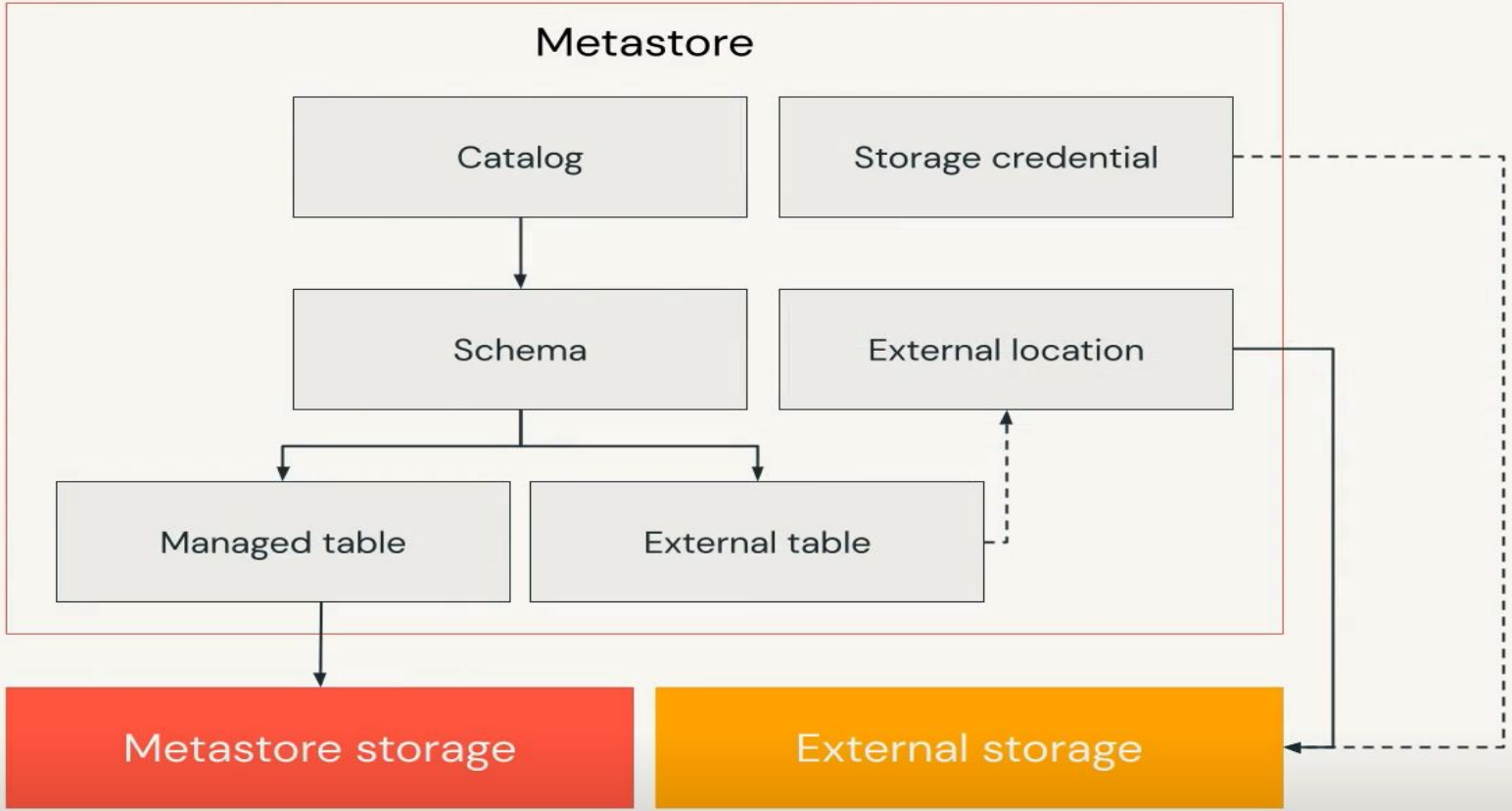
SQL Syntax for Access Control:

- Table Access:
  - `GRANT SELECT ON <catalog>.<schema>.<table> TO <group>;`
- Column Access:
  - Create dynamic views with conditional logic.
- Row Access:
  - Use dynamic views with conditional logic in `WHERE` clause.

Advanced Security:

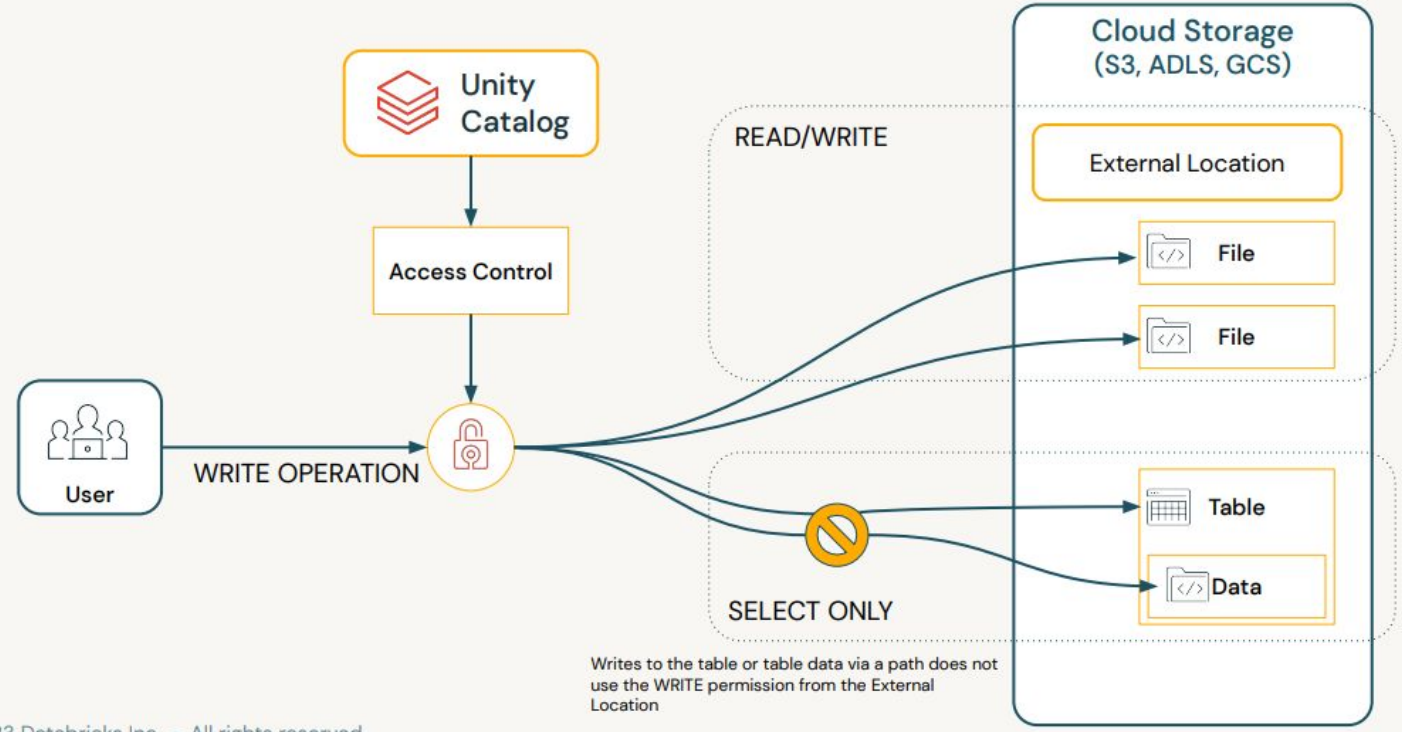
- Use row filters and column masks for extra granularity.

External Tables



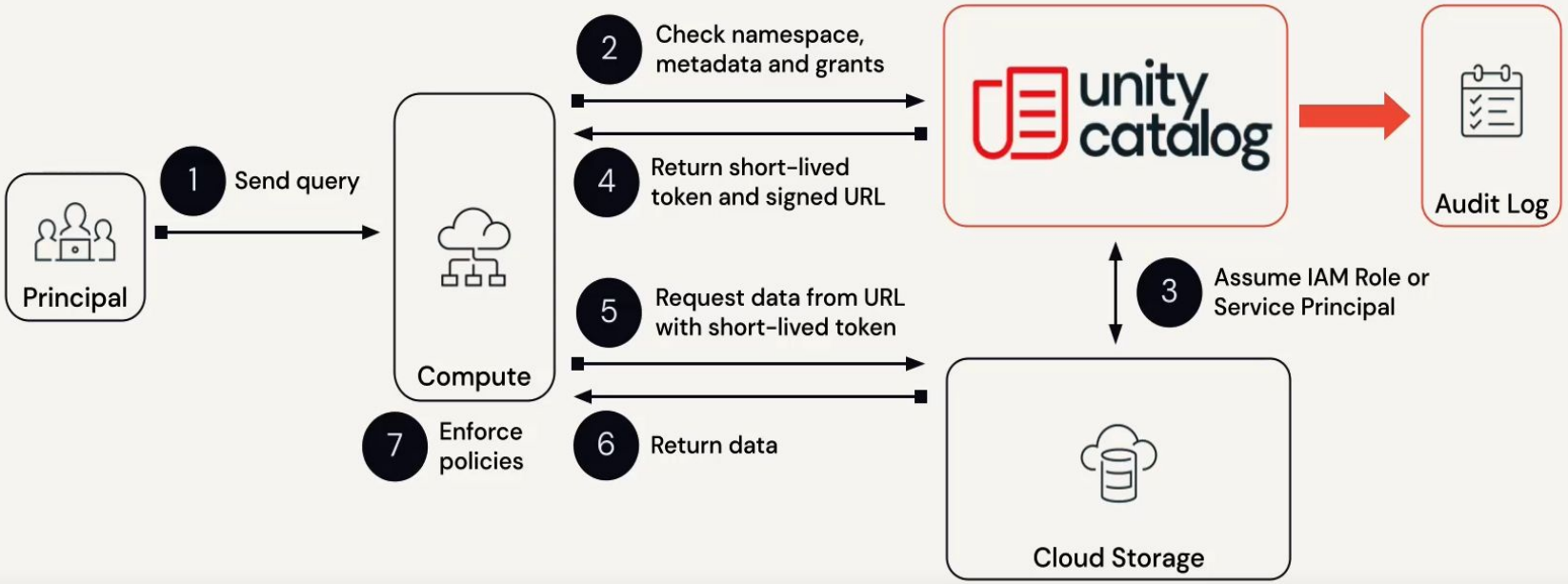
External Locations and External Tables

Access to an External Location does not imply access to External Tables



Security Model

Query Lifecycle



Row and column security

Enable fine-grained security

Use standard SQL functions to define row filters and column masks

Fine-grained access controls on rows and columns using SQL UDFs

Less overhead of creating views on the data for granular access controls

RBAC / ABAC coming in Q4/Q1

Public Preview in Q3 on AWS/GCP/Azure

```
// Row filtering

CREATE FUNCTION us_filter(region STRING)
RETURNS BOOLEAN
  RETURN if(is_member('admin'), true, region='US')

ALTER TABLE sales
SET ROW FILTER us_filter ON (region)
```

```
// Column masking

CREATE FUNCTION ssn_mask(ssn STRING)
RETURNS STRING
  RETURN if(is_member('admin'), ssn, '****')

ALTER TABLE users
ALTER COLUMN ssn SET MASK ssn_mask
```

Row Level Security and Column Level Masking

Provide fine grained differential access to datasets

In Preview

Only show specific rows

```
CREATE FUNCTION <name> ( <parameter_name >
<parameter_type> .. )
RETURN {filter clause whose output must be a boolean}


CREATE FUNCTION us_filter(region STRING)
RETURN IF(IS_MEMBER('admin'), true, region="US");

ALTER TABLE sales SET ROW FILTER us_filter ON region;
```

Test for group membership

Assign reusable filter to table

Specify filter predicates

Mask or redact sensitive columns

```
CREATE FUNCTION <name> (<parameter_name>,
<parameter_type>, [, <column>...])
RETURN {expression with the same type as the first
parameter}

CREATE FUNCTION ssn_mask(ssn STRING)
RETURN IF(IS_MEMBER('admin'), ssn, "****");

ALTER TABLE users ALTER COLUMN table_ssn SET MASK
ssn_mask;
```

Test for group membership

Assign reusable mask to column

Specify mask or function to mask

Discovery Tags

Semantic layer for your lakehouse

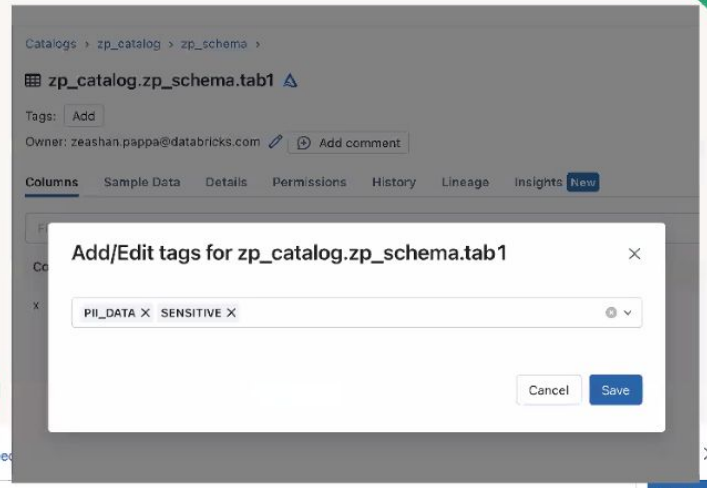
In Preview

Problem

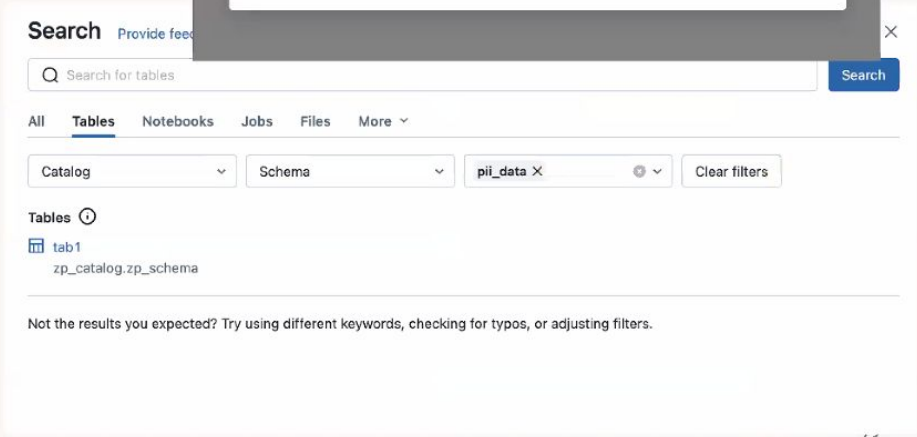
Searching for data assets in business terms or generally agreed upon taxonomies usually requires additional catalog tools.

Solution

- Discovery Tags allow you to tag Column, Table, Schema, Catalog objects in UC
- **Integrated search mechanism in UC allows you to search for objects by tag.**

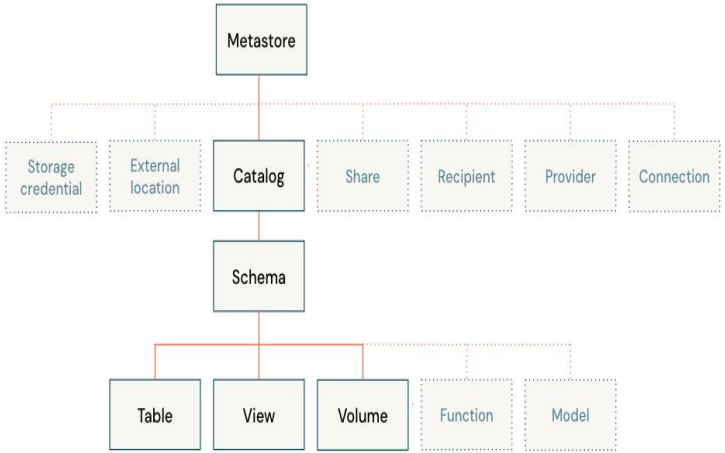


Tag your data



Search based on tags

Data Governance and Data Isolation



Unity Catalog provides a hierarchical model for data isolation using metastore, catalog, schema. Fine-grained access control is achieved via inherited permissions.

Building Blocks:

- Metastore: Top-level container for data objects in Unity Catalog.
  - Manages data assets and permissions.
  - Provides regional, not data, isolation.
- Catalog: Primary unit of data isolation.
  - Represents logical groupings of schemas.
  - Can be bound to specific workspaces (aka environments or business units).
- Schema (Database): More granular than catalogs.
  - Logical grouping of tables and views, and non-tabular data within a catalog
  - Organizes tables, views, volumes, functions, and ML models.
- Tables: Contain rows of data.
  - Managed by Unity Catalog or external platforms.
- Views: Read-only objects derived from tables and views.

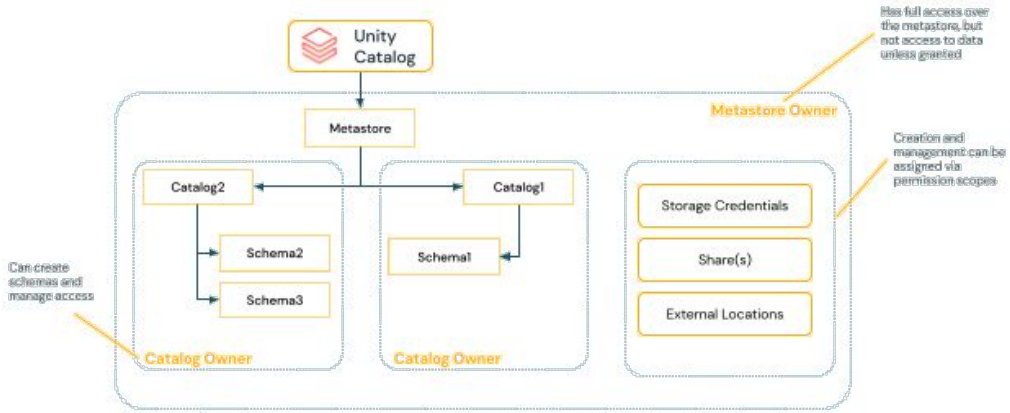
Other Components:

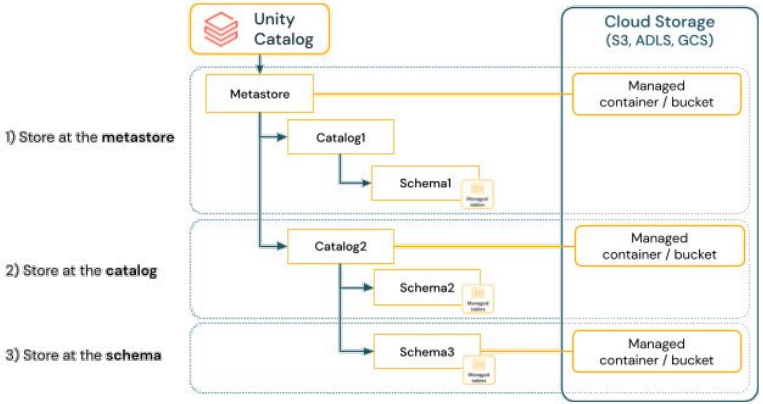
- Rows and Columns: Access controlled via dynamic views or filters.
- Volumes: Manage non-tabular data.

Key Points:

- Permissions are inherited through the hierarchy.
- Physical storage can be isolated at different levels.

Planning Data Isolation





Key Expectations:

- Access Control: Users access data based on specific rules.
- Management: Designated teams or people manage data.
- Physical Separation: Data is stored separately.
- Environment-specific Access: Data accessed only in designated settings.

Data Access:

- Strict requirements for data security (e.g., salaries, credit card info).
- Unity Catalog provides granular control for industry standards.

Governance Models:

- Centralized: Governance admins own metastore, grant/revoke permissions.
- Distributed: Catalog owners manage their own data domains independently.

Storage Options:

- Default storage is set at metastore creation.
- Option for separate storage locations at catalog and schema levels.

Workspace and Catalog Binding:

- Workspaces are primary data processing environments.
- Catalogs can be "bound" to specific workspaces for data isolation.

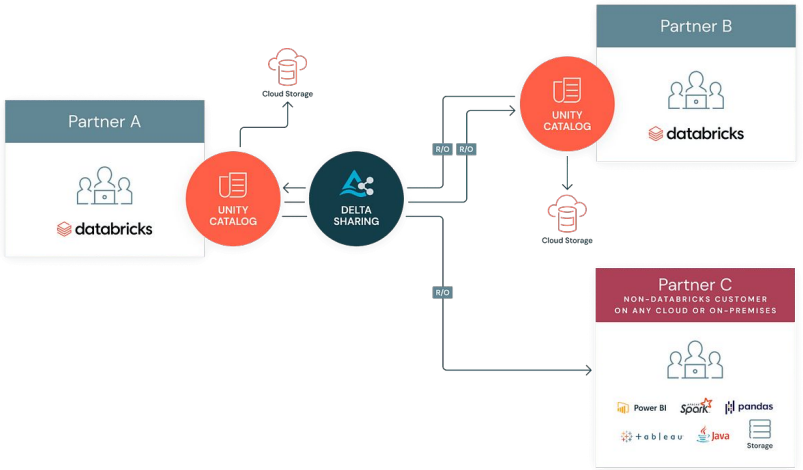
Metastore Configuration/Security

Key Points:

- Metastore is top level container for data objects and permissions

Setup Tips:

- One Per Region: One metastore for each region with Databricks workspaces.
- Data Sharing: Use Delta Sharing to share data between metastores.
- Root Storage: Ensure no direct user access to managed storage locations.
    - Metastore has default root storage for managed data
    - Dedicated bucket recommended.
    - Override options at catalog and schema levels.



Role & Privileges:

- Workspace Admins:
    - Can manage workspace operations but not data access.
    - Privileged role; distribute carefully.

Advanced:

- Workspace-Catalog Bindings:
    - Limit catalog access by workspace.
    - E.g., Restrict production data to production workspace.

External Locations and Storage Security

What Are They?

- External Locations: Access cloud storage on behalf of users. Paths to cloud storage + storage credentials.
- Storage Credential: Long-term cloud access credential (e.g., AWS IAM role).

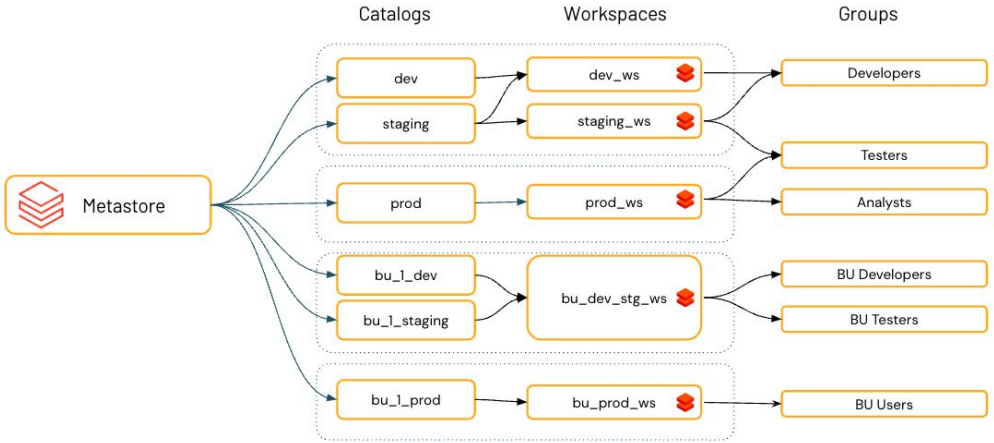
Use Cases:

- Register external tables and volumes in Unity Catalog.

Tips & Best Practices:

- Control & Auditability: External locations offer strong storage access management.
- Limit Direct Access: Fewer users should have direct access to external buckets.
- No DBFS Mounting: Avoid mounting the same storage accounts to DBFS and as external locations.
- Migrate Mounts: Move cloud storage mounts to Unity Catalog's external locations.

Organizing Data



Catalogs & Workspaces:

- Catalogs: Segregate data by environment, team, or unit.
- Workspace Binding: Bind catalogs to specific workspaces for data isolation.

Schema:

- Second layer in Unity Catalog.
- Organizes tables, views, and volumes.

Object Types:

- Managed Objects:
  - Lifecycle & file layout managed by Unity Catalog.
  - Ideal for governed locations without external management.
  - Always use Delta table format.
  - Simplicity
- External Objects:
  - Point to existing data
  - Not managed by Unity Catalog.
  - Useful for staging files produced by other systems.
  - Support multiple formats (Delta, Parquet, JSON, CSV).

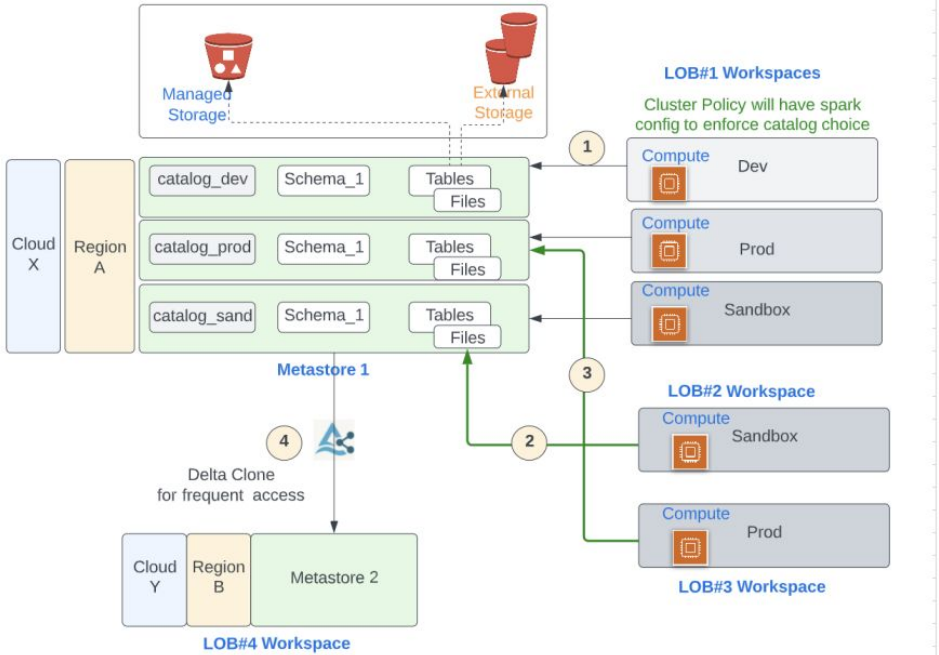
Storage Options:

- Optional managed storage at catalog or schema levels.

File Types:

- Managed & external volumes support structured, semi-structured, or unstructured data.

Scenario Examples



Scenario	Problem Statement
LOB#1	• Hosts separate workspaces for dev, prod and a shared sandbox environment • Each has a separate catalog. The underlying data can use either the managed storage or external storage locations. • Development workloads are promoted to prod by allowing compute clusters to automatically reference the relevant catalog as a cluster configuration parameter that can be enforced via cluster policy. These are different securables in the metastore and can have different privileges in dev/prod scope
LOB#2	• Hosts a sandbox environment that can access some assets from LOB#1 sandbox. This involves some users who also exist in LOB#1 and some new ones.
LOB#3	• Hosts a prod environment that uses some assets from LOB#1 prod to create derived products
LOB#4	• Is hosted in a different region/cloud and wishes to access some data produced by LOB#1

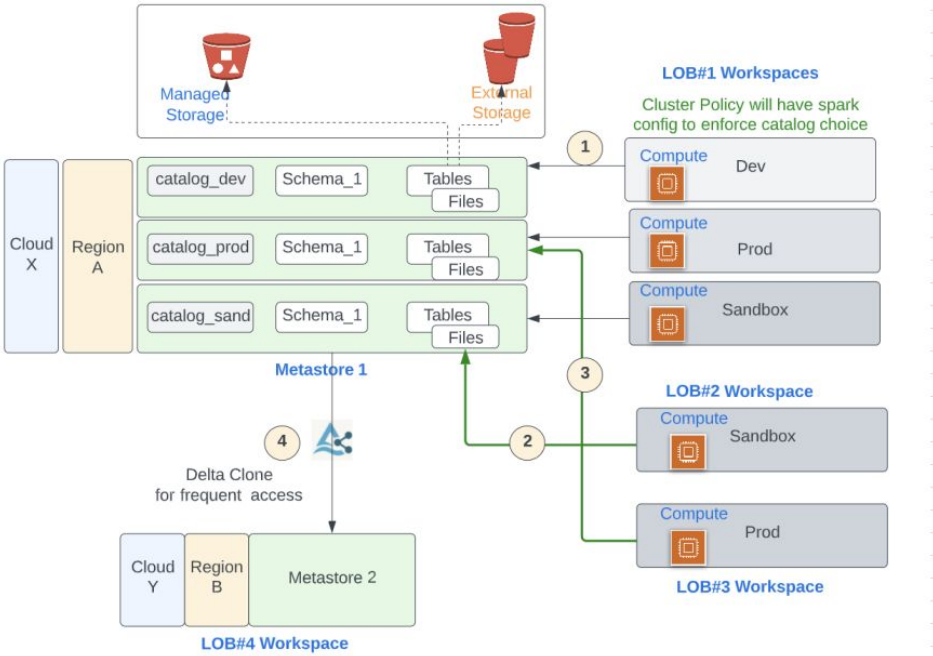
Scenario 1: LOB#1

```
-- Workspace Dev: Group1(Table1), Group2(Table2), Group3(Table3)
-- Assume a Metastore Admin has created CATALOG catalog_dev and schema1
GRANT USE CATALOG ON CATALOG catalog_dev to Group1;          -- repeat for Group2, Group3
GRANT USE SCHEMA ON SCHEMA catalog_dev.schema1 to Group1;    -- repeat for Group2, Group3
GRANT CREATE TABLE on SCHEMA catalog_dev.schema1 to Group1;  -- repeat for Group2, Group3

-- Assume Group1 creates Table1, Group2 creates Table2
-- By default, the person who creates becomes the owner
CREATE TABLE catalog_dev.schema1.Table1;                     -- Member of Group1 executes this
CREATE TABLE catalog_dev.schema1.Table2;                     -- Member of Group2 executes this

-- Either the table owner or someone in a more privileged role eg. schema/catalog/metastore owner
-- can perform GRANTS on their behalf or change ownership
GRANT SELECT on catalog_dev.schema1.Table1 to Group3;        -- Table Owner for Table1 runs this
GRANT SELECT on catalog_dev.schema1.Table2 to Group3;        -- Table Owner for Table2 runs this
```

```
-- Workspace Sandbox: Group1(Table1), Group2(Table2), Group3(Table3)
-- Scenario: Since this is sandbox we could give all privileges on the schema to all groups
GRANT USE CATALOG ON CATALOG catalog_sandbox to Group1;      -- repeat for Group2, Group3
GRANT ALL PRIVILEGES ON SCHEMA catalog_sandbox to Group1;    -- repeat for Group2, Group3

-- Assume a new Table2 is created
CREATE TABLE catalog_sandbox.schema1.Table2;                 -- Member of Group2 executes this
-- We will come back to the above table in the LOB2 scenario
```

```
-- Workspace Prod: ServicePrincipal
-- Scenario: A Production job run by a Service Principal reads from Table1 and writes to Table2
GRANT USE CATALOG ON CATALOG catalog_prod to SP;
GRANT SELECT on catalog_prod.schema_1.table1 to SP;
GRANT MODIFY on catalog_prod.schema_1.table2 to SP;
```

Scenario Examples

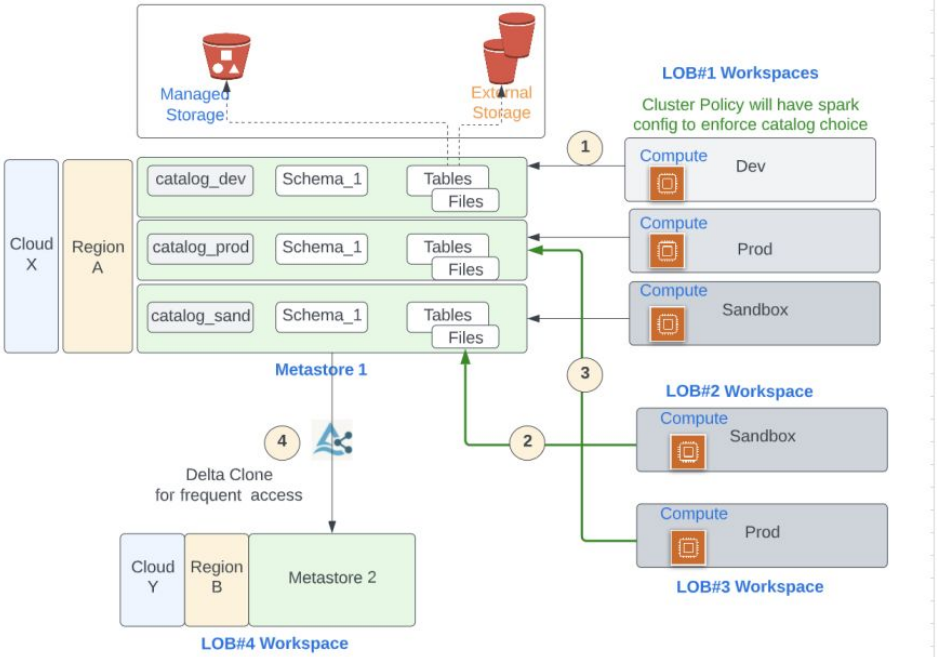


Scenario	Problem Statement
LOB#1	• Hosts separate workspaces for dev, prod and a shared sandbox environment • Each has a separate catalog. The underlying data can use either the managed storage or external storage locations. • Development workloads are promoted to prod by allowing compute clusters to automatically reference the relevant catalog as a cluster configuration parameter that can be enforced via cluster policy. These are different securables in the metastore and can have different privileges in dev/prod scope
LOB#2	• Hosts a sandbox environment that can access some assets from LOB#1 sandbox. This involves some users who also exist in LOB#1 and some new ones.
LOB#3	• Hosts a prod environment that uses some assets from LOB#1 prod to create derived products
LOB#4	• Is hosted in a different region/cloud and wishes to access some data produced by LOB#1

Scenario 2: LOB#2

```
-- Scenario: Group2 and Group4 in LOB2's Sandbox Workspace want to access the same data as was prepared in the LOB1's catalog_sandbox
-- No additional grants are needed for Group2 as they were already granted
-- Members of Group2 can access the same data that they could from previous LOB#1 Sandbox WS
-- We only need to provide the new access to Group4
GRANT USE CATALOG ON CATALOG catalog_sandbox to Group4;
GRANT USE SCHEMA on SCHEMA catalog_sandbox.schema1 to Group4;
GRANT SELECT, MODIFY on catalog_sandbox.schema1.Table2 to Group4;
-- Now Group4 can work on this table from this workspace (and any other workspace that you may connect to the same metastore)
```

Scenario Examples



Scenario	Problem Statement
LOB#1	• Hosts separate workspaces for dev, prod and a shared sandbox environment • Each has a separate catalog. The underlying data can use either the managed storage or external storage locations. • Development workloads are promoted to prod by allowing compute clusters to automatically reference the relevant catalog as a cluster configuration parameter that can be enforced via cluster policy. These are different securables in the metastore and can have different privileges in dev/prod scope
LOB#2	• Hosts a sandbox environment that can access some assets from LOB#1 sandbox. This involves some users who also exist in LOB#1 and some new ones.
LOB#3	• Hosts a prod environment that uses some assets from LOB#1 prod to create derived products
LOB#4	• Is hosted in a different region/cloud and wishes to access some data produced by LOB#1

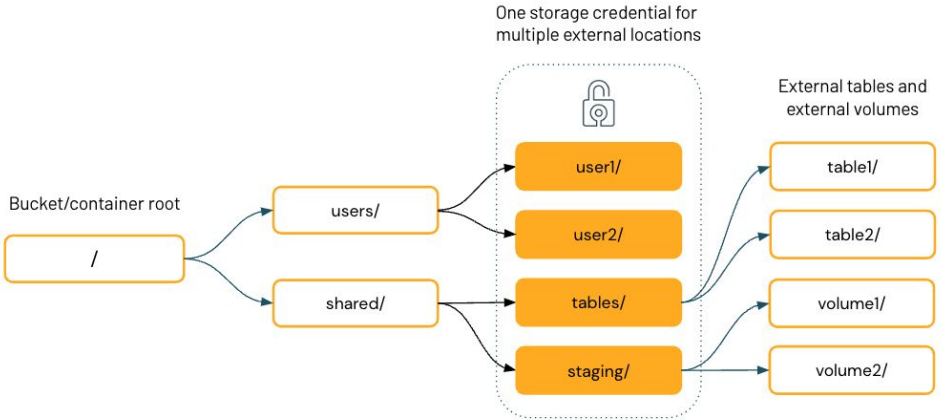
Scenario 3: LOB#3

```
-- Scenario: Workspace Prod has a new Group5 and they need access to LOB1's catalog_prod to create a derived data product
GRANT USE SCHEMA on SCHEMA catalog_prod.schema1 to Group5;
GRANT SELECT on catalog_prod.schema1.Table2 to Group5;
-- Now Group5 can read from this table from this workspace (and any other workspace that you may connect to the same metastore)
-- An upcoming feature of workspace-catalog bindings can be used to restrict the workspaces that specific catalogs can be accessed from thus
-- ensuring that Prod data can only be accessed from Prod workspaces, as an example
```

Scenario 4: LOB#4

```
-- Scenario: Share Data with Metastore in a different region or cloud
CREATE SHARE IF NOT EXISTS ShareToLob4;
ALTER SHARE ShareToLob4  ADD TABLE catalog_sandbox.schema1.Table2;
CREATE RECIPIENT IF NOT EXISTS RecipientLob4;
GRANT SELECT ON SHARE ShareToLob4 TO RECIPIENT RecipientLob4;
-- Now Recipient Lob4 which is a metastore on a different cloud/region can read from this share, which has 1 table.
-- The share can be altered at any time to add or remove tables
-- The access to the share at the recipient can be similarly controlled via ACLs in the recipient metastore
```

External Locations, Tables, Volumes



External Locations:

- Purpose: Enable Unity Catalog to manage cloud storage.
- Access Control: Limit user/group access to specific cloud storage directories.

Recommendations for External Locations:

- Admin Control: Only trusted admins should create external locations.
- Granular Permissions: Use tables and volumes for more specific access control.
- Avoid General Permissions: Don't grant general READ/WRITE permissions to end-users.

Use Cases for External Locations:

- Register external tables and volumes.
- Explore existing cloud files.
- Set as managed storage for catalogs and schemas.

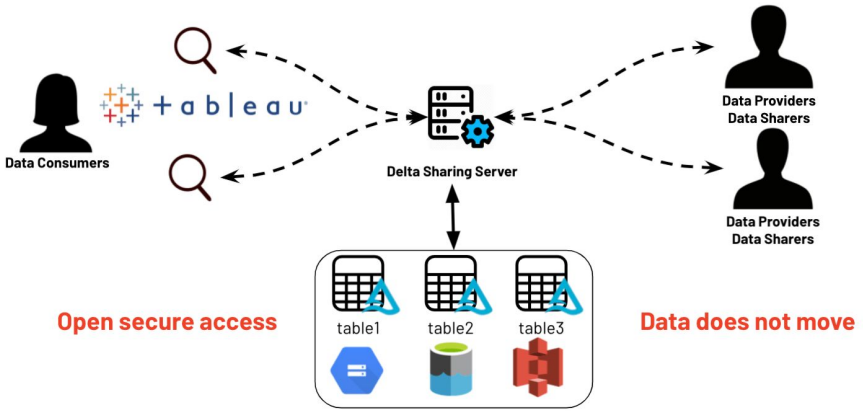
External Volumes:

- Use For:
  - ETL pipelines and data engineering.
  - Staging locations for ingestion.
  - Data storage for EDA and ML tasks.
- Tips:
  - Create from one external location within one schema.
  - Ideal for ingestion use-cases.

External Tables:

- Use For: Normal querying patterns on cloud data.
- Recommendations:
  - Create from one external location within one schema.
  - Avoid multi-metastore registration due to consistency risks.

Securing Delta Sharing



What is Delta Sharing?

- An open protocol by Databricks for secure data sharing.
- Works across different organizations and computing platforms.
- Unity Catalog runs a Delta Sharing server when enabled.

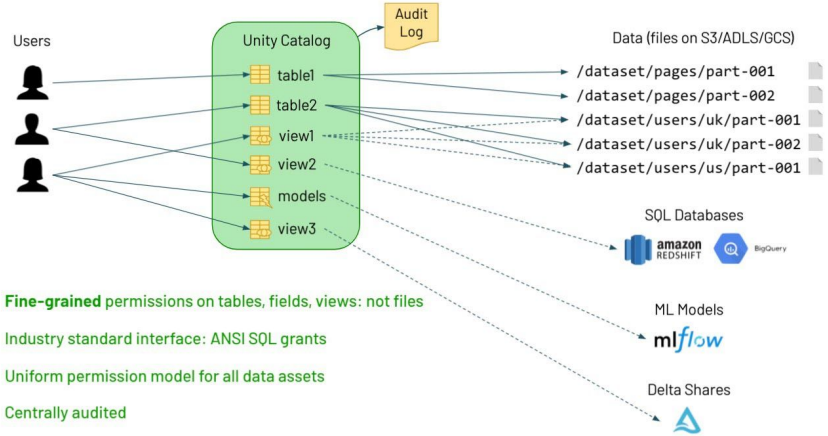
Databricks-to-Databricks Sharing:

- Allows cross-metastore sharing.
- Tables appear as read-only objects in the consuming metastore.
- Can be granted access like any other object within Unity Catalog.

Important Notes on Access Control:

- Limited to one metastore.
- Grants from the source metastore do not apply to the destination.
- Destination must set its own access controls.

Auditing Access



- Who accessed this table?
- Which users accessed a table within the last day?
- Which tables did a user access?
- View all permissions changes
- View the most recently run notebook commands

- Complete governance requires auditing data access.
- Key components of a complete data governance solution.

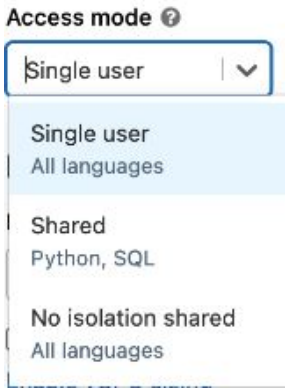
Unity Catalog:

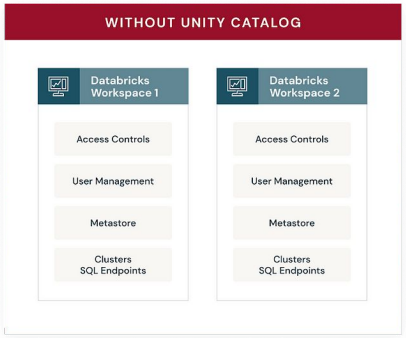
- Captures audit logs for actions performed against the metastore.
- Logs integrated into Databricks audit logs.

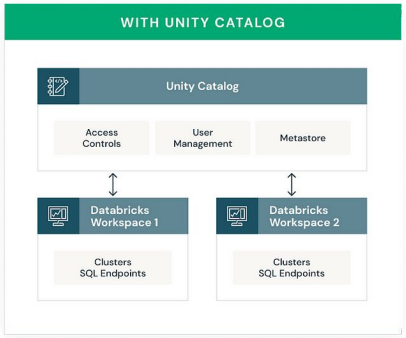
Accessing Audit Logs:

- Audit Log Delivery
- Available via system tables.

Cluster Configurations







Cluster Policies:

- Purpose: Limit cluster configurations based on rules.
- Benefits:
  - Simplifies cluster creation for users.
  - Controls costs by setting per-cluster maximums.

Unity Catalog & Security:

- Access Modes: Ensures strong isolation and integrity of access controls.
- Default: Secure, requires proper access mode for data access.

Access Modes:

- Shared Access Mode: For sharing a cluster.
- Single User Access Mode: For automated jobs and ML workloads.
- No Isolation: Legacy (No UC Access)

Use Cluster Policy Definitions (to enforce UC-Only Clusters):

- Shared Access Mode:
  - `access_mode: "USER_ISOLATION"`
- Single User Access Mode:
  - `access_mode: "SINGLE_USER"`

Next Steps

1. All users for all customers have been synced to the account and deduped automatically.
2. Identify the groups that has to be created at the account level
   a. Table ACLs
   b. Volumes ACLS
   c. Row Level Filtering, Column Level Filtering
   d. Access to UC functionality (Delta Sharing)
3. Create the groups at the account
4. Enable SCIM or other automation option (recommended)
5. Repoint local workspace assets to the account level group (programatically)

Challenges with Setting up Groups

What works what doesn't

1. Groups are set at the account level (not Workspace or Metastore)
2. Groups require a unique name
3. Workspace groups with the same name have to be deduped
4. Nested Groups are not supported by the Azure SCIM enterprise app

Group Management with "UC by Default"

- Distributed approach to group management
- Group can be created at the workspace
- Group owners can manage their groups

Decisions to be made for Project Upgrade Plan & Solution Architecture Documents

- Create Solution Architecture Document
  - Session 1 Feedback
    - Metastore Configuration
    - Storage Configuration
    - Workspaces
  - Session 2 Feedback
    - Metastore Catalog Permissions
    - IAM Permissions to DBR Objects
    - Service Principals
    - User Groups
    - Admin Groups