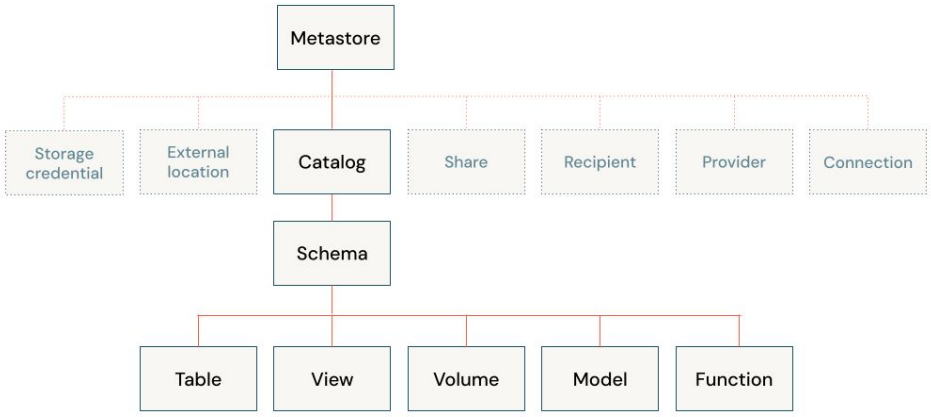
databricks

Unity Catalog

LECTURE

Data Isolation

The Data Hierarchy in Unity Catalog

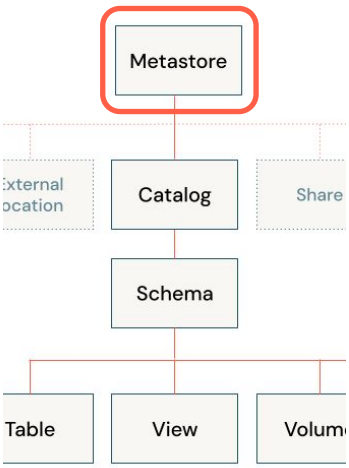


Note: Some securable objects are grayed out to emphasize the hierarchy of objects managed under catalogs

- About Data Isolation is important to understand and consider Unity Catalog's data hierarchy, which is very easy to follow with a metastore and its association with a three-level namespace to organize your data assets.
- In this session, we'll review the "Metastore", Catalogs and Volumes.

Metastores

- Manage data assets (tables, views, and volumes) and the permissions that govern access to them.
- Admins create **one metastore per region**
- Metastores are mapped to one or more **workspaces within same region**
- Metastores provide regional isolation but are **not intended as units of data isolation**

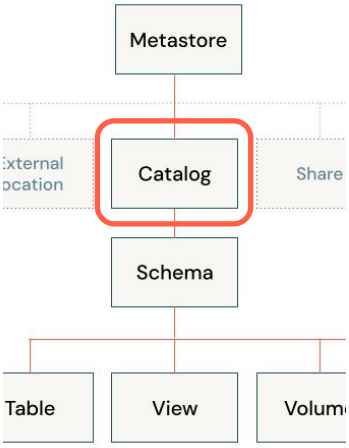


Metastore

- It's the top-level container for metadata in Databricks Unity Catalog and It organizes data objects hierarchically following the three-level namespace.
- Account administrators can create one metastore per region and assign it to multiple workspaces in that region.
- Just in case a workspace is the working environment for a a group of users.
- Metastores offer regional isolation by default, with physical storage for each metastore typically separated from others in the same account
- But Data isolation should begin at the catalog level, which is the highest level in the data hierarchy (catalog > schema > table/view/volume).

Catalogs

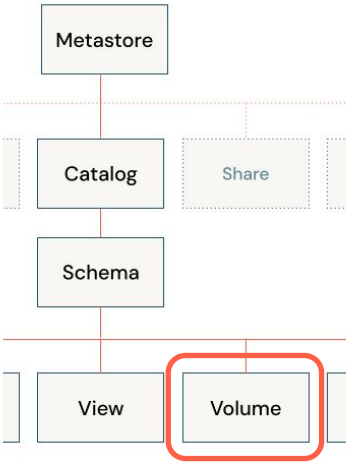
- Intended as the **primary unit of data isolation**
- Often mirror organizational units or software development lifecycle scopes
- Can be stored at the metastore level, or **stored separately from the rest of the parent metastore (preferred)**
- Can be bound to specific workspaces
- Ideal spot to set **inherited permissions**



- As mentioned Catalogs are intended as the primary unit of data isolation.
- They often mirror organizational units or software development lifecycle scopes, such as separate catalogs for production and development data.
- Catalogs can be stored at the metastore level or separately from the parent metastore, with separate storage being the preferred option.
- They can be bound to specific workspaces, ensuring certain types of data are processed only in designated environments.
- Catalogs serve as an ideal spot to set inherited permissions, allowing for efficient and granular access control.

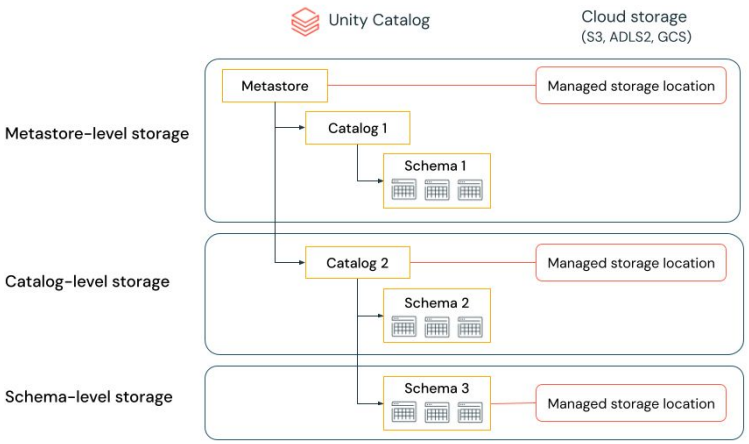
Volumes

- Used for **non-tabular data**
- Any data - structured, semi-structured and unstructured - can be stored here
- Perfect for **libraries, configurations, checkpoint** folders
- Data in Volumes **cannot be registered** as Tables



- Volumes can store any type of data, including structured, semi-structured, and unstructured formats.
- Volumes offer capabilities for accessing, storing, governing, and organizing files. Such as libraries, configurations, and checkpoint folders.
- They provide governance over non-tabular datasets, complementing the governance that tables offer for tabular data.
- Something important to understand is that the data stored in volumes cannot be registered as tables or handled as a table.
- They can be either managed (stored in Unity Catalog-managed locations) or external (registered against directories in external locations).

Physically Separating Data



An organization can require that data of certain types be stored within specific accounts or buckets in their cloud tenant.

Unity Catalog gives the ability to configure storage locations at the metastore, catalog, or schema level to satisfy such requirements.

Further,

Unity Catalog gives you the ability to choose between centralized and distributed governance models.

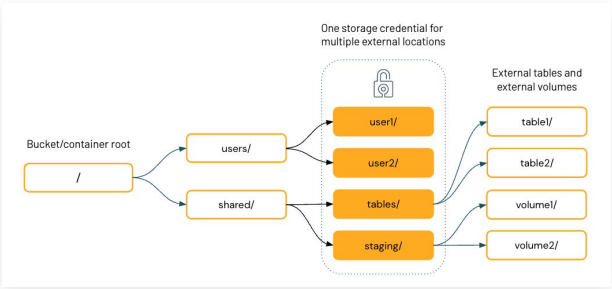
In the centralized governance model, your governance administrators are owners of the metastore and can take ownership of any object and grant and revoke permissions.

In a distributed governance model, the catalog or a set of catalogs is the data domain. The owner of that catalog can create and own all assets and manage governance within that domain. The owners of any given domain can operate independently of the owners of other domains.

Regardless of whether you choose the metastore or catalogs as your data domain, Databricks strongly recommends that you set a group as the metastore admin or catalog owner.

External Locations and Storage Credentials

- External locations are defined as a **path to cloud storage, combined with a storage credential** that can be used to access that location.



- Allow Unity Catalog to **read and write data on your cloud tenant**

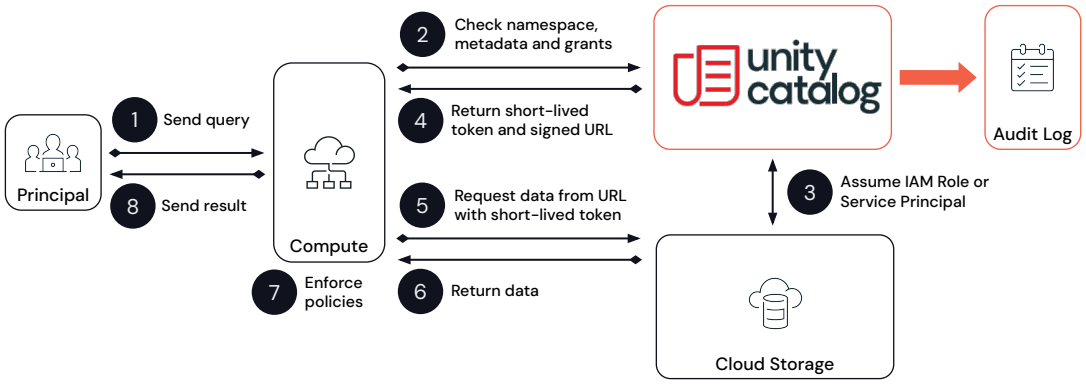
- Use external locations to **register external tables and external volumes** in Unity Catalog.

External locations and storage credentials in Databricks Unity Catalog play a crucial role in data isolation:

- External locations associate Unity Catalog storage credentials with cloud object storage containers. They allow Unity Catalog to read and write data on your cloud tenant on behalf of users.
- For enhanced data isolation, external locations and storage credentials can be bound to specific workspaces.
- External locations provide strong control and auditability of storage access.
- To prevent bypassing Unity Catalog access controls, limit direct user access to containers used as external locations.

Accessing Data Securely

The Unity Catalog Security Model



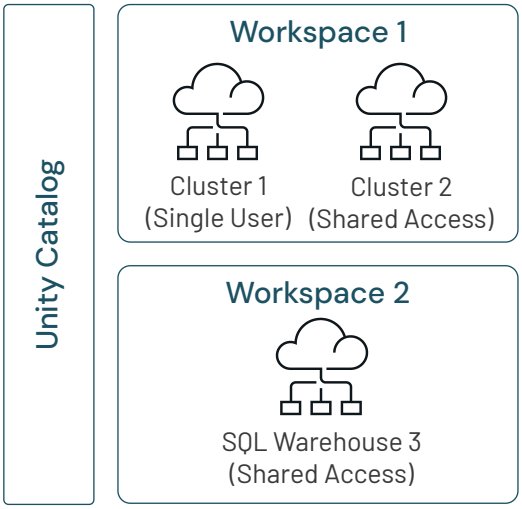
Having discussed all the key concepts related to Unity Catalog, let's take a final look at it's security model in action. Let's begin by touring the lifecyle of a query to see how Unity Catalog provides access control in a secure yet performant way.

(SLIDE CONTAINS ANIMATIONS; ONE FOR EACH STEP BELOW)

1. [CLICK]The story begins with a principal issuing a query. Queries can be issued through all-purpose clusters, for cases when users are running Python or SQL workloads interactively. In the case of a job or pipeline running as a service principal, this would typically run through a job cluster. Alternatively, data analysts may issue queries in Databricks SQL through a SQL warehouse, or the query could be originating from a BI tool connected to a SQL warehouse. In any case, the applicable compute resource begins processing the query.
2. [CLICK]The request is dispatched to Unity catalog, which in turn logs the request and validates the query against all security constraints defined within the metastore to which the compute resource is associated.
3. [CLICK]For each object referenced in the query, Unity Catalog assumes the appropriate cloud credential governing that object, as provided by a cloud administrator. For managed tables, this would be the cloud storage associated with the metastore. For files or external tables, this would be an external location governed by a storage credential.
4. [CLICK]Again, for each object referenced in the query, Unity Catalog generates a scoped temporary token to enable a client to access the data directly from storage and return that token, along with an access URL. This

1. allows the cluster or SQL warehouse to access data directly but securely.
2. [CLICK]The cluster or SQL warehouse requests data directly from cloud storage using the URL and token passed back from Unity Catalog.
3. [CLICK]Data is transferred back from cloud storage. This request process is repeated for each object referenced by the query.
4. [CLICK]With access to data at the partition level, last-mile row or column based filtering is applied on the cluster or SQL warehouse.
5. [CLICK]Finally, filtered result is passed back to the caller.

Data and Workload Isolation



Easily isolate based on your needs

1. **Single-user clusters** provide data protection
2. **Multi-user clusters** that safely support users with different privileges, such as Shared access mode or SQL Warehouses.
3. **Multiple workspaces** isolate groups who won't collaborate

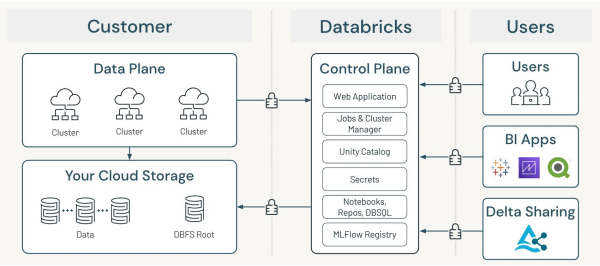
Another way to consider to isolate your data is by leveraging proper cluster profile and the use of workspaces.

Here we have an image that provides two workspaces where the first has two clusters one for a single user that may work on a POC and other for a shared access for your team to work on active projects, meanwhile in the second workspace has a SQL Warehouse cluster that allows collaboration across your team.

Encryption by Default

All of the following are encrypted by default

- All traffic between control plane and data plane
- All traffic between the user and the control plane
- All storage in the Databricks account
- All EBS volumes
- All traffic to AWS APIs



You can rest assure as Databricks employs encryption measures to protect data in all transmisions:

- Data in transit: All traffic between users and the control plane, control plane and compute plane, and to AWS APIs is encrypted using TLS/SSL protocols.
- Data at rest: AES-256 encryption is used for data stored in Azure Storage and Amazon EBS volumes.
- Control plane data: Envelope encryption is used, where the data encryption key (DEK) is encrypted with a customer-managed key (CMK) and then re-encrypted with a Databricks-managed key.

https://www.databricks.com/trust/security-features/data-protection-with-customer-managed-keys

Manage Access to PII

- Control access to storage locations with cloud permissions
- Limit human access to raw data
- Unity Catalog row and column filtering
- Pseudonymize records on ingestion
- Use table ACLs to manage user permissions
- Configure dynamic views for data redaction
- Remove identifying details from demographic views

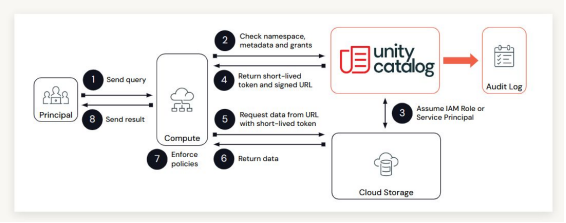
While Databricks has extensive access control lists, physically separating private data into storage containers with limited access throughout the organization is an important extra step. Even if you're not specifically relying on cloud identity access management for granting data access in Databricks, having correctly set permissions in the cloud is critical.

Because clusters and attached storage volumes are ephemeral, cached data will not persist after a job completes. If you're saving personally identifiable information, removing natural keys that might link back to the user identity limits the number of places you'll need to delete data and guarantee it's no longer identifiable.

When making data available for analysts, views can be used to either dynamically mask or de-identify columns through aggregation.

New Best Practices

1. **Avoid direct access to object stores**
   a. Less keys => Less leaks
   b. No more credentials in code or secret scopes
   c. UC mediates all access to data
   d. UC maintains audit trail of all access requests
   e. Only infra admins will have object store access



2. **Avoid having data assets in hive metastore**
   a. Hive metastore are not secure enough. All new assets go into UC.
3. **Leverage Workspace–local model registries**
   a. All Model assets will now be stored with schemas in UC.
4. **Avoid DBFS usage**
   a. All unstructured data (checkpoints, libraries, config files etc) should be placed in Volumes

Now let's go over the new “best practices” suggested by Databricks:

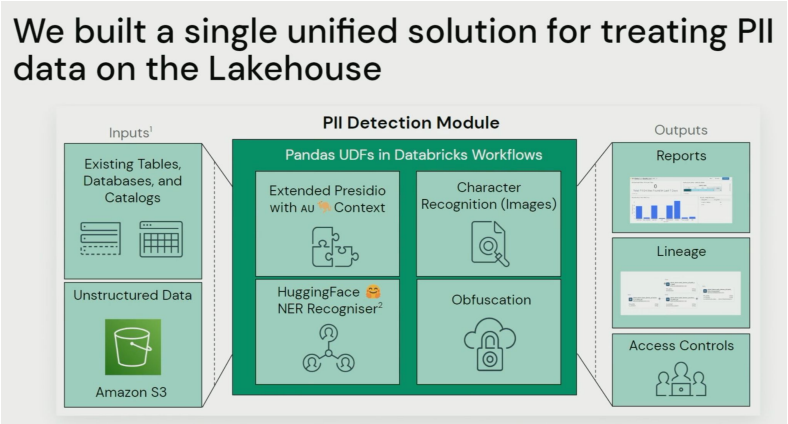
1) Avoid direct access to object stores. Unity Catalog should mediate all data access, maintaining a comprehensive audit trail of all access requests. This centralized approach ensures better control and visibility over data access.
2) Minimize key usage: Reducing the number of keys in circulation decreases the risk of security breaches. Unity Catalog's centralized access control eliminates the need for multiple access keys.
3) Eliminate credentials in code or secret scopes: With Unity Catalog, sensitive credentials are no longer stored in code or secret scopes. Instead, leverage Databricks Secrets.
4) Migrate from Hive metastore: Hive metastores are considered less secure. All new data assets should be created and managed within the Unity Catalog.
5) For Model assets, leverage Unity Catalog: Store all model assets within schemas in Unity Catalog instead of workspace-local model registries. This approach provides better governance and sharing capabilities across workspaces.
6) Avoid DBFS usage: Databricks File System (DBFS) should be avoided in Unity Catalog-enabled workspaces. Instead, use Volume to store unstructured data such as checkpoints, libraries, and configuration files.

Here is a link for more best practices

https://docs.databricks.com/en/data-governance/unity-catalog/best-practices.html

Customer Use Case – SEEK

PII Detection at Scale on the Lakehouse



Here, we have a Success Case from SEEK, Australia's largest online employment marketplace. This marketplace processes millions of resumes containing sensitive candidate information.

- Their goal is to create a near real-time data surveillance system that exposes gaps, delivers dashboards, and triggers tickets for data custodians to address issues.
- SEEK developed a custom framework using HuggingFace transformers to detect and anonymize PII in both structured and unstructured data
- Their PII detection module includes:
  - Customized recognizers for specific contexts
  - Optical Character Recognition (OCR) for detecting PII in uploaded documents like passports
  - Integration with MLflow for model versioning and API creation
  - Real-time alerts to prevent users from submitting sensitive information
- The system uses Unity Catalog for data lineage tracking and access control enforcement

Their approach aims to transform data governance from a manual process to an AI-driven, automated system that enhances privacy protection and risk management, so check them out.