Last updated on ***Dec 30, 2025***

What is Unity Catalog?

This article introduces Unity Catalog, a unified governance solution for data and AI assets on Databricks. It explains key concepts and gives an overview of how to use Unity Catalog to govern data.

Unity Catalog is also available as an open-source implementation. See the announcement blog and the public Unity Catalog GitHub repo.

Overview of Unity Catalog

Unity Catalog is a centralized data catalog that provides access control, auditing, lineage, quality monitoring, and data discovery capabilities across Databricks workspaces.

Key features of Unity Catalog include:

- **Define once, secure everywhere**: Unity Catalog offers a single place to administer data access policies that apply across all workspaces in a region.
- **Standards-compliant security model**: Unity Catalog's security model is based on standard ANSI SQL and allows administrators to grant permissions in their existing data lake using familiar syntax.
- **Built-in auditing and lineage**: Unity Catalog automatically captures user-level audit logs that record access to your data. Unity Catalog also captures lineage data that tracks how data assets are created and used across all languages.
- **Data discovery**: Unity Catalog lets you tag and document data assets, and provides a search interface to help data consumers find data.
- **System tables**: Unity Catalog lets you easily access and query your account's operational data, including audit logs, billable usage, and lineage.

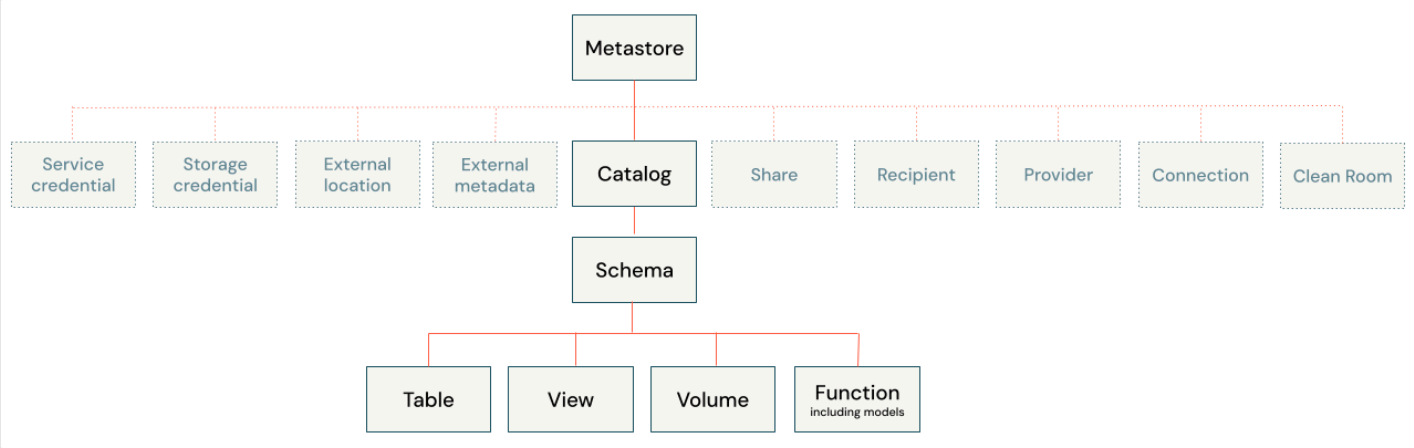
Metastore

The metastore is the top-level container for metadata in Unity Catalog. It registers metadata about data and AI assets and the permissions that govern access to them. For a workspace to use Unity Catalog, it must have a Unity Catalog metastore attached. You should have one metastore for each region in which you have workspaces.

Unlike Hive metastore, the Unity Catalog metastore is not a service boundary: it runs in a multi-tenant environment and represents a logical boundary for the segregation of data by region for a given Databricks account.

The Unity Catalog object model

In a Unity Catalog metastore, the three-level database object hierarchy consists of catalogs that contain schemas, which in turn contain data and AI objects, like tables and models. This hierarchy is represented as a three-level namespace (`catalog.schema.table-etc`) when you reference tables, views, volumes, models, and functions.



Level one:

- **Catalogs** are used to organize your data assets and are typically used as the top level in your data isolation scheme. Catalogs often mirror organizational units or software development lifecycle scopes. See What are catalogs in Databricks?.
- **Non-data securable objects**, such as storage credentials and external locations, are used to manage your data governance model in Unity Catalog. These also live directly under the metastore. They are described in more detail in Securable objects that Unity Catalog uses to manage access to external data sources.

Level two:

- **Schemas** (also known as databases) contain tables, views, volumes, AI models, and functions. Schemas organize data and AI assets into logical categories that are more granular than catalogs. Typically a schema represents a single use case, project, or team sandbox. See What are schemas in Databricks?.

Level three:

- **Tables** are collections of data organized by rows and columns. Tables can be either *managed*, with Unity Catalog managing the full lifecycle of the table, or *external*, with Unity Catalog managing access to the data from within Databricks, but not managing access to the data in cloud storage from other clients. See Databricks tables and Managed versus external tables and volumes.
- **Views** are saved queries against one or more tables. See What is a view?.
- **Volumes** represent logical volumes of data in cloud object storage. You can use volumes to store, organize, and access files in any format, including structured, semi-structured, and unstructured data. Typically they are used for non-tabular data. Volumes can be either *managed*, with Unity Catalog managing the full lifecycle and layout of the data in storage, or *external*, with Unity Catalog managing access to the data from within Databricks, but not managing access to the data in cloud storage from other clients. See What are Unity Catalog volumes? and Managed versus external tables and volumes.
- **Functions** are units of saved logic that return a scalar value or set of rows. See User-defined functions (UDFs) in Unity Catalog.
- **Models** are AI models packaged with MLflow and registered in Unity Catalog as functions. See Manage model lifecycle in Unity Catalog.

Securable objects that Unity Catalog uses to manage access to external data sources

In addition to the database objects and AI assets that are contained in schemas, Unity Catalog also uses the following securable objects to manage access to cloud storage and other external data sources and services:

- **Storage credentials**, which encapsulate a long-term cloud credential that provides access to cloud storage. See Overview of storage credentials.

- **External locations**, which reference both a cloud storage path and the storage credential required to access it. External locations can be used to create external tables or to assign a **managed storage location** for managed tables and volumes. See Overview of external locations, Cloud storage and data isolation, and Specify a managed storage location in Unity Catalog.
- **Connections**, which represent credentials that give read-only access to an external database in a database system like MySQL using Lakehouse Federation. See What is Lakehouse Federation?.
- **Service credentials**, which encapsulate a long-term cloud credential that provides access to an external service. See Create service credentials.

Securable objects that Unity Catalog uses to manage access to shared assets

Unity Catalog uses the following securable objects to manage data and AI asset sharing across metastore or organizational boundaries:

- **Clean rooms**, which represent a Databricks-managed environment where multiple participants can collaborate on projects without sharing underlying data with each other. See What is Databricks Clean Rooms?.
- **Shares**, which are Delta Sharing objects that represent a read-only collection of data and AI assets that a data provider shares with one or more recipients.
- **Recipients**, which are Delta Sharing objects that represent an entity that receives shares from a data provider.
- **Providers**, which are Delta Sharing objects that represent an entity that shares data with a recipient.

For more information about the Delta Sharing securable objects, see What is Delta Sharing?.

Admin roles

The following Databricks admin roles have many Unity Catalog privileges by default:

- Account admins: can create metastores, link workspaces to metastores, add users, and assign privileges on metastores.
- Workspace admins: can add users to a workspace, and manage many workspace-specific objects like jobs and notebooks. Depending on the workspace, workspace admins can also have many privileges on the metastore that is attached to the workspace.
- *Metastore admins*: This optional role is required if you want to manage table and volume storage at the metastore level. It is also convenient if you want to manage data centrally across multiple workspaces in a region.

For more information, see Admin privileges in Unity Catalog.

Granting and revoking access to securable objects

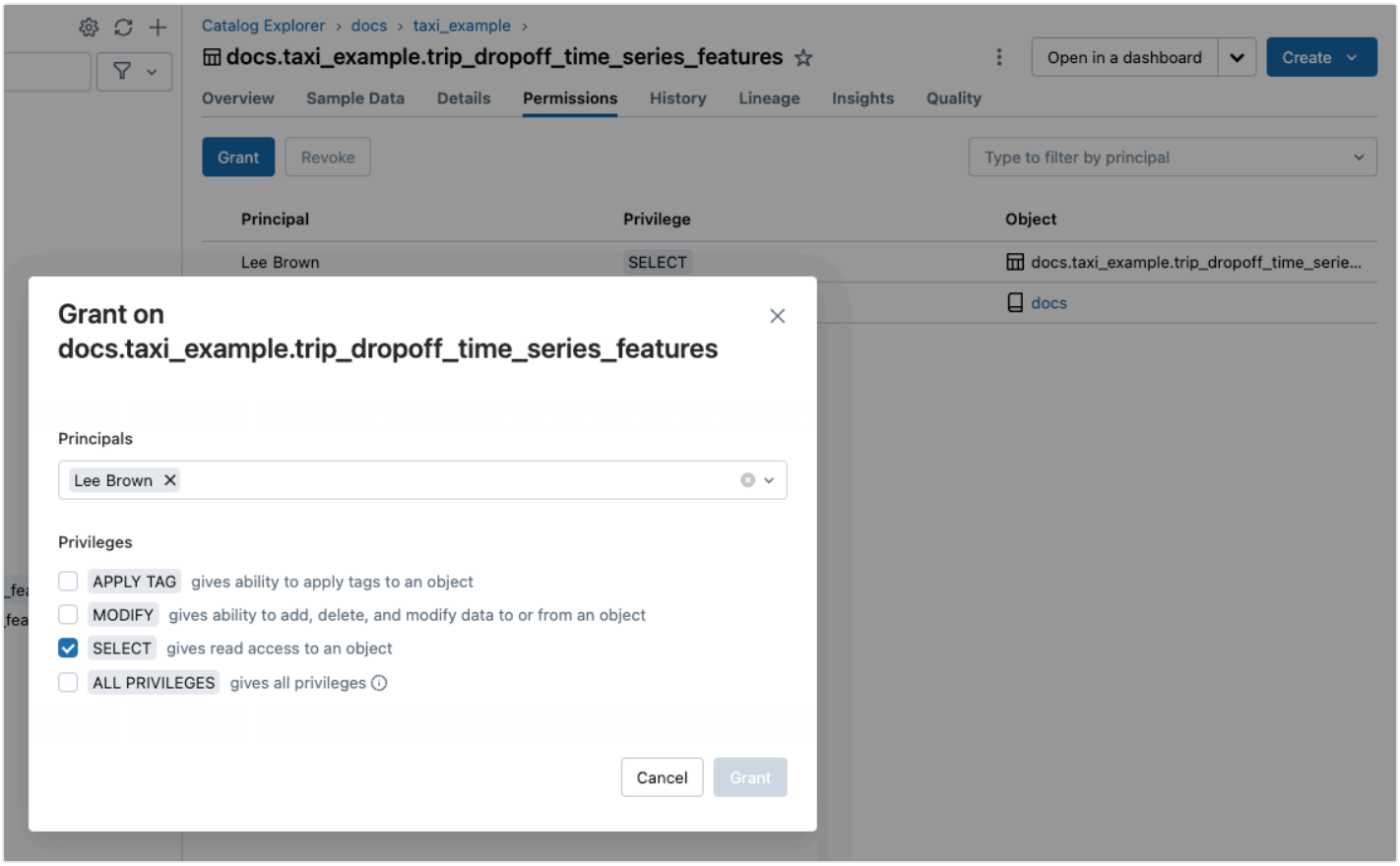
Privileged users can grant and revoke access to securable objects at any level in the hierarchy, including the metastore itself. Access to an object implicitly grants the same access to all children of that object, unless access is revoked.

You can use typical ANSI SQL commands to grant and revoke access to objects in Unity Catalog. For example:

SQL

```
GRANT CREATE TABLE ON SCHEMA mycatalog.myschema TO `finance-team`;
```

You can also use Catalog Explorer, the Databricks CLI, and REST APIs to manage object permissions.



Metastore admins, owners of an object, and users with the `MANAGE privilege` on an object can grant and revoke access. To learn how to manage privileges in Unity Catalog, see Manage privileges in Unity Catalog.

Default access to database objects in Unity Catalog

Unity Catalog operates on the principle of least privilege, where users have the minimum access they need to perform their required tasks. When a workspace is created, non-admin users have access only to the automatically-provisioned **Workspace catalog**, which makes this catalog a convenient place for users to try out the process of creating and accessing database objects in Unity Catalog. See Workspace catalog privileges.

Working with database objects in Unity Catalog

Working with database objects in Unity Catalog is very similar to working with database objects that are registered in a Hive metastore, with the exception that a Hive metastore doesn't include catalogs in the object namespace. You can use familiar ANSI syntax to create database objects,

manage database objects, manage permissions, and work with data in Unity Catalog. You can also create database objects, manage database objects, and manage permissions on database objects using the Catalog Explorer UI.

For more information, see Database objects in Databricks.

Managed versus external tables and volumes

Tables and volumes can be managed or external.

- **Managed tables** are fully managed by Unity Catalog, which means that Unity Catalog manages both the governance and the underlying data files for each managed table. Managed tables are stored in a Unity Catalog-managed location in your cloud storage. Managed tables always use the Delta Lake format. You can store managed tables at the metastore, catalog, or schema levels.
- **External tables** are tables whose access from Databricks is managed by Unity Catalog, but whose data lifecycle and file layout are managed using your cloud provider and other data platforms. Typically you use external tables to register large amounts of your existing data in Databricks, or if you also require write access to the data using tools outside of Databricks. External tables are supported in multiple data formats. Once an external table is registered in a Unity Catalog metastore, you can manage and audit Databricks access to it---and work with it---just like you can with managed tables.
- **Managed volumes** are fully managed by Unity Catalog, which means that Unity Catalog manages access to the volume's storage location in your cloud provider account. When you create a managed volume, it is automatically stored in the *managed storage location* assigned to the containing schema.
- **External volumes** represent existing data in storage locations that are managed outside of Databricks, but registered in Unity Catalog to control and audit access from within Databricks. When you create an external volume in Databricks, you specify its location, which must be on a path that is defined in a Unity Catalog *external location*.

Databricks recommends managed tables and volumes for most use-cases, because they allow you to take full advantage of Unity Catalog governance capabilities and performance

optimizations. For information about typical use-cases for external tables and volumes, see Managed and external tables and Managed and external volumes.

See also:

- Unity Catalog managed tables in Databricks for Delta Lake and Apache Iceberg
- Work with external tables
- Managed versus external volumes.

Cloud storage and data isolation

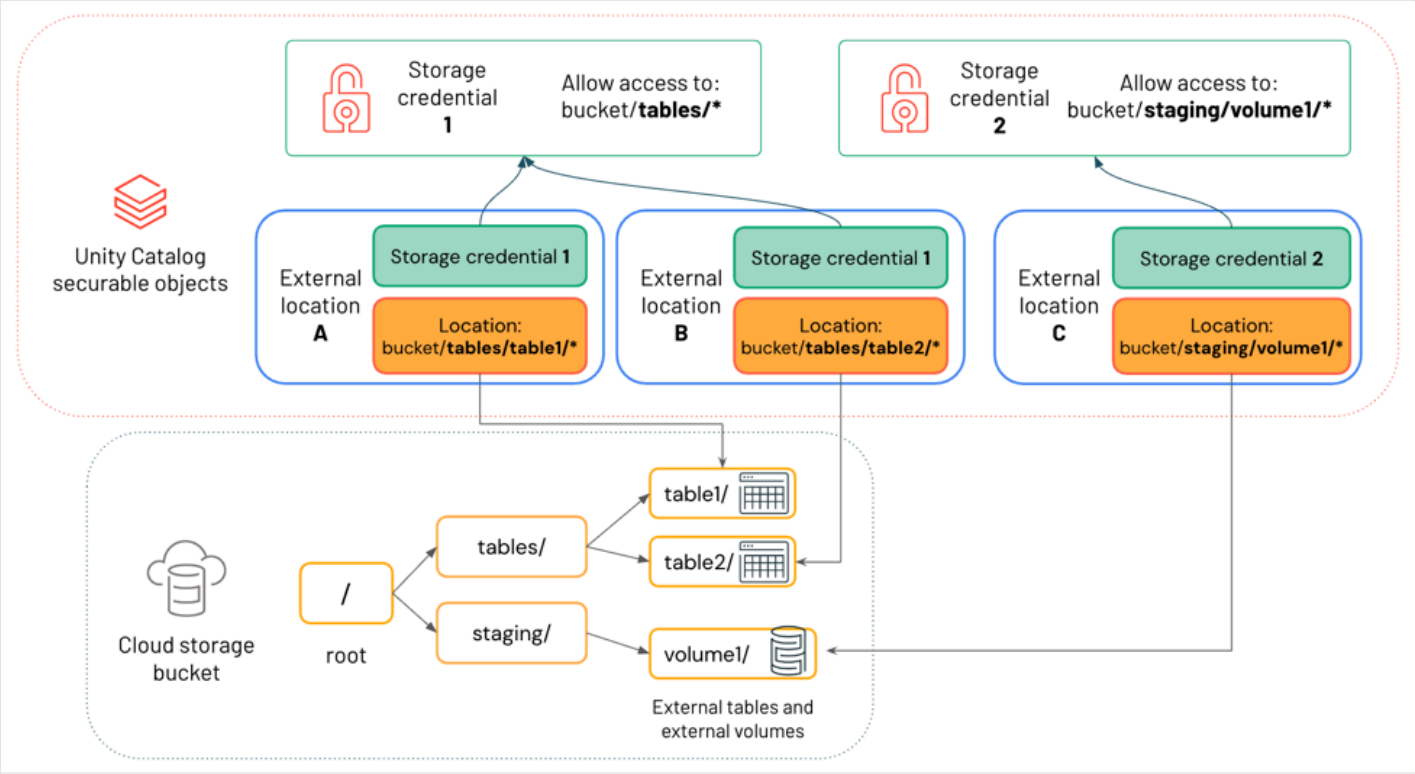
Unity Catalog uses cloud storage in two primary ways:

- *Managed storage*: default locations for managed tables and managed volumes (unstructured, non-tabular data) that you create in Databricks. These managed storage locations can be defined at the metastore, catalog, or schema level. You create managed storage locations in your cloud provider, but their lifecycle is fully managed by Unity Catalog.
- Storage locations where external tables and volumes are stored. These are tables and volumes whose access from Databricks is managed by Unity Catalog, but whose data lifecycle and file layout are managed using your cloud provider and other data platforms. Typically you use external tables or volumes to register large amounts of your existing data in Databricks, or if you also require write access to the data using tools outside of Databricks.

Governing access to cloud storage using external locations

Both managed storage locations and storage locations where external tables and volumes are stored use *external location* securable objects to manage access from Databricks. External location objects reference a cloud storage path and the *storage credential* required to access it. Storage credentials are themselves Unity Catalog securable objects that register the credentials required to access a particular storage path. Together, these securables ensure that access to storage is controlled and tracked by Unity Catalog.

The diagram below shows how external locations reference storage credentials and cloud storage locations.



In this diagram:

- Each external location references a storage credential and a cloud storage location.
- Multiple external locations can reference the same storage credential. **Storage credential 1** grants access to everything under the path `bucket/tables/*`, so both **External location A** and **External location B** reference it.

For more information, see How does Unity Catalog govern access to cloud storage?.

Managed storage location hierarchy

The level at which you define managed storage in Unity Catalog depends on your preferred data isolation model. Your organization may require that certain types of data be stored within specific accounts or buckets in your cloud tenant.

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

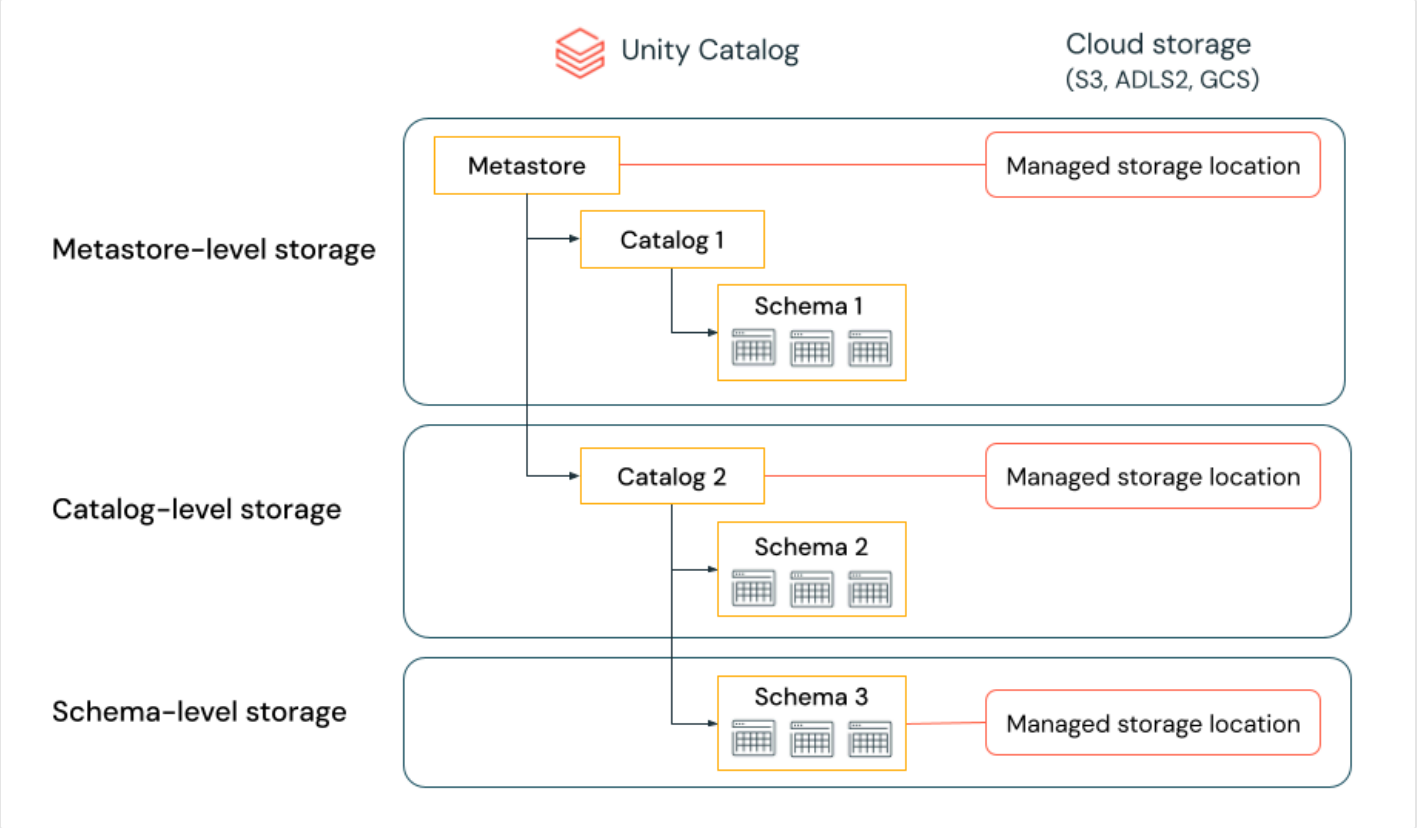
For example, let's say your organization has a company compliance policy that requires production data relating to human resources to reside in the bucket s3://mycompany-hr-prod. In Unity Catalog, you can achieve this requirement by setting a location on a catalog level, creating a catalog called, for example `hr_prod`, and assigning the location s3://mycompany-hr-prod/unity-catalog to it. This means that managed tables or volumes created in the `hr_prod` catalog (for example, using `CREATE TABLE hr_prod.default.table …`) store their data in s3://mycompany-hr-prod/unity-catalog. Optionally, you can choose to provide schema-level locations to organize data within the `hr_prod catalog` at a more granular level.

If storage isolation is not required for some catalogs, you can optionally set a storage location at the metastore level. This location serves as a default location for managed tables and volumes in catalogs and schemas that don't have assigned storage. Typically, however, Databricks recommends that you assign separate managed storage locations for each catalog.

The system evaluates the hierarchy of storage locations from schema to catalog to metastore.

For example, if a table `myCatalog.mySchema.myTable` is created in `my-region-metastore`, the table storage location is determined according to the following rule:

1. If a location has been provided for `mySchema`, it will be stored there.
2. If not, and a location has been provided on `myCatalog`, it will be stored there.
3. Finally, if no location has been provided on `myCatalog`, it will be stored in the location associated with the `my-region-metastore`.



For more information, see Specify a managed storage location in Unity Catalog.

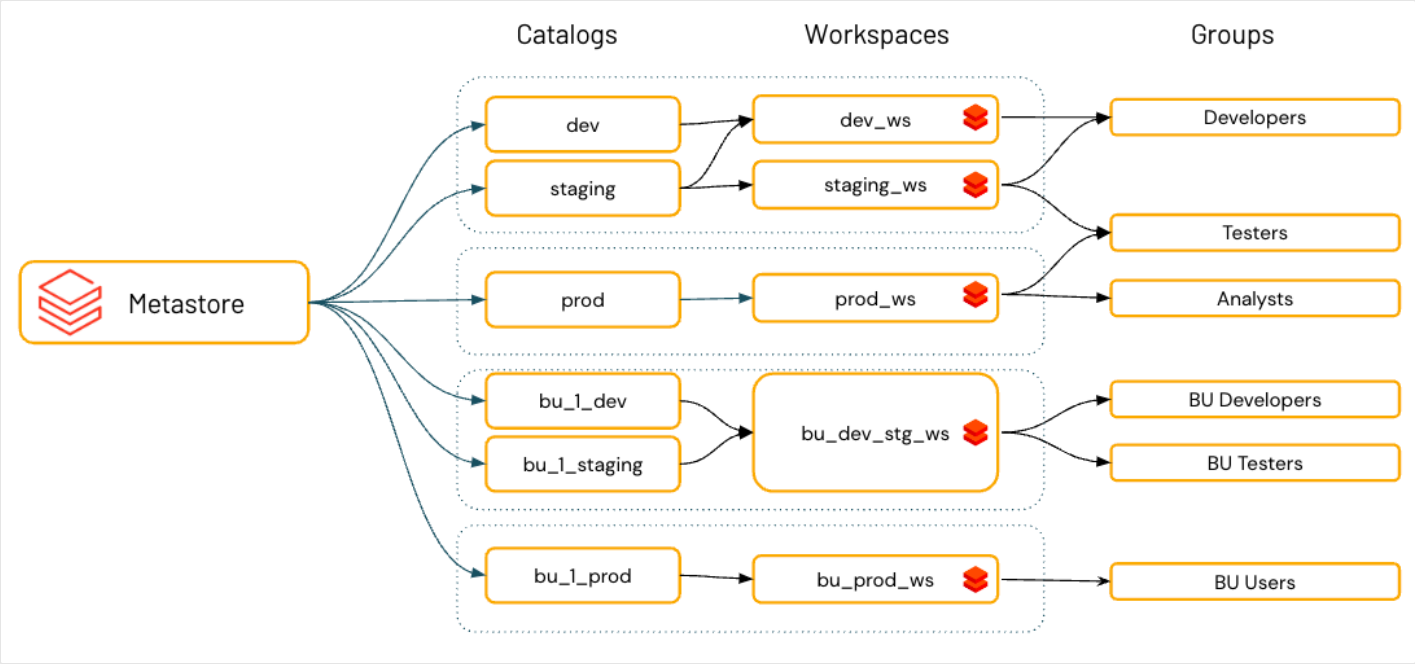
Environment isolation using workspace-catalog binding

By default, catalog owners (and metastore admins, if they are defined for the account) can make a catalog accessible to users in multiple workspaces attached to the same Unity Catalog metastore.

Organizational and compliance requirements often specify that you keep certain data, like personal data, accessible only in certain environments. You may also want to keep production data isolated from development environments or ensure that certain data sets and domains are never joined together.

In Databricks, the workspace is the primary data processing environment, and catalogs are the primary data domain. Unity Catalog lets metastore admins, catalog owners, and users with the `MANAGE` permission assign, or "bind," catalogs to specific workspaces. These environment-aware bindings give you the ability to ensure that only certain catalogs are available within a workspace, regardless of the specific privileges on data objects granted to a user. If you use workspaces to isolate user data access, however, you might want to limit catalog access to

specific workspaces in your account, to ensure that certain kinds of data are processed only in those workspaces. You might want separate production and development workspaces, for example, or a separate workspace for processing personal data. This is known as *workspace-catalog binding*. See Limit catalog access to specific workspaces.



> **NOTE**
>
> For increased data isolation, you can also bind cloud storage access and cloud service access to specific workspaces. See Assign a storage credential to specific workspaces, Assign an external location to specific workspaces, and (Optional) Assign a service credential to specific workspaces.

How do I set up Unity Catalog for my organization?

To use Unity Catalog, your Databricks workspace must be enabled for Unity Catalog, which means that the workspace is attached to a Unity Catalog metastore.

How does a workspace get attached to a metastore? It depends on the account and the workspace:

- Typically, when you create a Databricks workspace in a region for the first time, the metastore is created automatically and attached to the workspace.
- For some older accounts, an account admin must create the metastore and assign the workspaces in that region to the metastore. For instructions, see Create a Unity Catalog metastore.
- If an account already has a metastore assigned for a region, an account admin can decide whether to attach the metastore automatically to all new workspaces in that region. See Enable a metastore to be automatically assigned to new workspaces.

Whether or not your workspace was enabled for Unity Catalog automatically, the following steps are also required to get started with Unity Catalog:

- Create catalogs and schemas to contain database objects like tables and volumes.
- Create managed storage locations to store the managed tables and volumes in these catalogs and schemas.
- Grant user access to catalogs, schemas, and database objects.

Workspaces that are automatically enabled for Unity Catalog provision a *workspace catalog* with broad privileges granted to all workspace users. This catalog is a convenient starting point for trying out Unity Catalog.

For detailed setup instructions, see Get started with Unity Catalog.

Upgrading an existing workspace to Unity Catalog

To learn how to upgrade a non-Unity Catalog workspace to Unity Catalog, see Upgrade a Databricks workspaces to Unity Catalog.

Unity Catalog requirements and restrictions

Unity Catalog requires specific types of compute and file formats, described below. Also listed below are some Databricks features that are not fully supported in Unity Catalog on all

Databricks Runtime versions.

Region support

All regions support Unity Catalog. For details, see Databricks clouds and regions.

Compute requirements

Unity Catalog is supported on clusters that run Databricks Runtime 11.3 LTS or above. Unity Catalog is supported by default on all SQL warehouse compute versions.

Clusters running on earlier versions of Databricks Runtime do not provide support for all Unity Catalog GA features and functionality.

To access data in Unity Catalog, clusters must be configured with the correct *access mode*. Unity Catalog is secure by default. If a cluster is not configured with standard or dedicated access mode, the cluster can't access data in Unity Catalog. See Access modes.

For detailed information about Unity Catalog functionality changes in each Databricks Runtime version, see the release notes.

File format support

Unity Catalog supports the following table formats:

- Managed tables must use the `delta` table format.
- External tables can use `delta`, `CSV`, `JSON`, `avro`, `parquet`, `ORC`, or `text`.

Securable object naming requirements

The following limitations apply for all object names in Unity Catalog:

- Object names cannot exceed 255 characters.
- The following special characters are not allowed:
  - Period (`.`)
  - Space ( )

    - Forward slash (`/`)
    - All ASCII control characters (00-1F hex)
    - The DELETE character (7F hex)
- Unity Catalog stores all object names as lowercase.
- When referencing UC names in SQL, you must use backticks to escape names that contain special characters such as hyphens (`-`).

> **NOTE**
>
> Column names can use special characters, but the name must be escaped with backticks in all SQL statements if special characters are used. Unity Catalog preserves column name casing, but queries against Unity Catalog tables are case-insensitive.

Limitations

Unity Catalog has the following limitations. Some of these are specific to older Databricks Runtime versions and compute access modes.

Structured Streaming workloads have additional limitations, depending on Databricks Runtime and access mode. See Standard compute requirements and limitations and Dedicated compute requirements and limitations.

Databricks releases new functionality that shrinks this list regularly.

- Groups that were previously created in a workspace (that is, workspace-level groups) cannot be used in Unity Catalog `GRANT` statements. This is to ensure a consistent view of groups that can span across workspaces. To use groups in `GRANT` statements, create your groups at the account level and update any automation for principal or group management (such as SCIM, Okta and Microsoft Entra ID connectors, and Terraform) to reference account endpoints instead of workspace endpoints. See Group sources.

- Workloads in R do not support the use of dynamic views for row-level or column-level security on compute running Databricks Runtime 15.3 and below.

    - Use a dedicated compute resource running Databricks Runtime 15.4 LTS or above for workloads in R that query dynamic views. Such workloads also require a workspace

that is enabled for serverless compute. For details, see Fine-grained access control on dedicated compute.

- Shallow clones are unsupported in Unity Catalog on compute running Databricks Runtime 12.2 LTS and below. You can use shallow clones to create managed tables on Databricks Runtime 13.3 LTS and above. You cannot use them to create external tables, regardless of Databricks Runtime version. See Shallow clone for Unity Catalog tables.

- Bucketing is not supported for Unity Catalog tables. If you run commands that try to create a bucketed table in Unity Catalog, it will throw an exception.

- Writing to the same path or Delta Lake table from workspaces in multiple regions can lead to unreliable performance if some clusters access Unity Catalog and others do not.

- Manipulating partitions for external tables using commands like `ALTER TABLE ADD PARTITION` requires partition metadata logging to be enabled. See Partition discovery for external tables.

- When using overwrite mode for tables not in Delta format, the user must have the CREATE TABLE privilege on the parent schema and must be the owner of the existing object OR have the MODIFY privilege on the object.

- Python UDFs are not supported in Databricks Runtime 12.2 LTS and below. This includes UDAFs, UDTFs, and Pandas on Spark (`applyInPandas` and `mapInPandas`). Python scalar UDFs are supported in Databricks Runtime 13.3 LTS and above.

- Scala UDFs are not supported in Databricks Runtime 14.1 and below on compute with standard access mode. Scalar UDFs are supported in Databricks Runtime 14.2 and above on compute with standard access mode.

- Standard Scala thread pools are not supported. Instead, use the special thread pools in `org.apache.spark.util.ThreadUtils`, for example, `org.apache.spark.util.ThreadUtils.newDaemonFixedThreadPool`. However, the following thread pools in `ThreadUtils` are not supported: `ThreadUtils.newForkJoinPool` and any `ScheduledExecutorService` thread pool.

Models registered in Unity Catalog have additional limitations. See Limitations.

Resource quotas

Unity Catalog enforces resource quotas on all securable objects. These quotas are listed in Resource limits. If you expect to exceed these resource limits, contact your Databricks account team.

You can monitor your quota usage using the Unity Catalog resource quotas APIs. See Monitor your usage of Unity Catalog resource quotas.

Additional resources

- Data governance with Databricks
- Upgrade a Databricks workspaces to Unity Catalog
- Unity Catalog best practices
- Resolve storage path conflicts