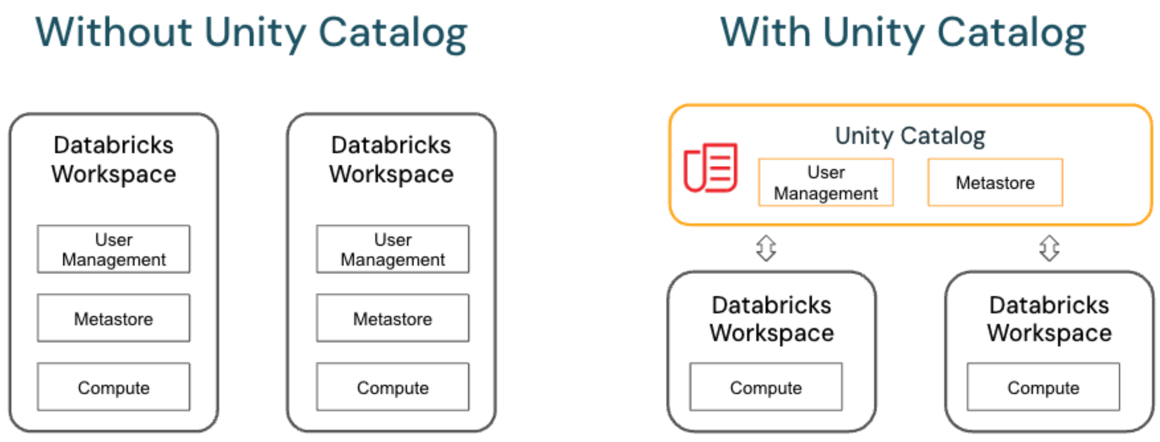
Unity Catalog

Overview of Unity Catalog

Unity Catalog provides centralized access control, auditing, lineage, and data discovery capabilities across Azure 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.
- **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, at the level of catalogs, databases (also called schemas), tables, and views.
- **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 (Public Preview)**: Unity Catalog lets you easily access and query your account's operational data, including audit logs, billable usage, and lineage.

How does Unity Catalog govern access to data and AI assets in cloud object storage?

Databricks recommends that you configure all access to cloud object storage using Unity Catalog.

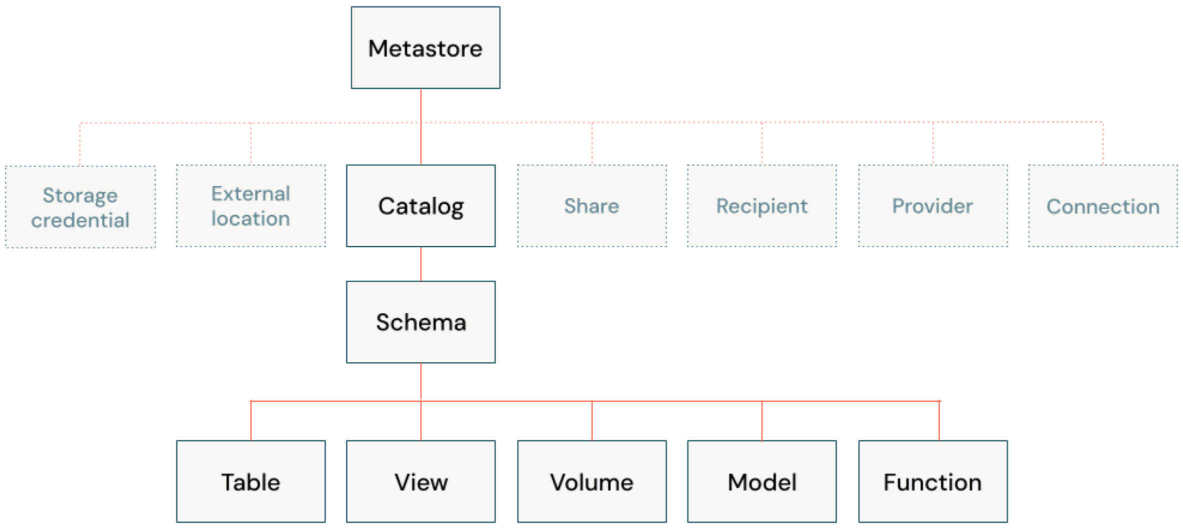
Unity Catalog introduces the following concepts to manage relationships between data in Azure Databricks and cloud object storage:

- **Storage credentials** encapsulate a long-term cloud credential that provides access to cloud storage. For example, an Azure managed identity that can access an Azure Data Lake Storage Gen2 container or a Cloudflare R2 API token.
- **External locations** contain a reference to a storage credential and a cloud storage path.
- **Managed storage locations** associate a storage location in an Azure Data Lake Storage Gen2 container or Cloudflare R2 bucket in your own cloud storage account with a metastore, catalog, or schema. Managed storage locations are used as default storage location for managed tables and managed volumes.
- **Volumes** provide access to non-tabular data stored in cloud object storage.
- **Tables** provide access to tabular data stored in cloud object storage.

The Unity Catalog object model

In Unity Catalog, the hierarchy of primary data objects flows from metastore to table or volume:

- **Metastore**: The top-level container for metadata. Each metastore exposes a three-level namespace (`catalog.schema.table`) that organizes your data.
- **Catalog**: The first layer of the object hierarchy, used to organize your data assets.
- **Schema**: Also known as databases, schemas are the second layer of the object hierarchy and contain tables and views.
- **Tables, views, and volumes**: At the lowest level in the data object hierarchy are tables, views, and volumes. Volumes provide governance for non-tabular data.
- **Models**: Although they are not, strictly speaking, data assets, registered models can also be managed in Unity Catalog and reside at the lowest level in the object hierarchy.



This is a simplified view of securable Unity Catalog objects.

You reference all data in Unity Catalog using a three-level namespace: `catalog.schema.asset`, where `asset` can be a table, view, volume, or model.

Metastores

A metastore is the top-level container of objects in Unity Catalog. It registers metadata about data and AI assets and the permissions that govern access to them. Azure Databricks account admins should create one metastore for each region in which they operate and assign them to Azure Databricks workspaces in the same region. For a workspace to use Unity Catalog, it must have a Unity Catalog metastore attached.

A metastore can optionally be configured with a managed storage location in an Azure Data Lake Storage Gen2 container or Cloudflare R2 bucket in your own cloud storage account.

Catalogs

A catalog is the first layer of Unity Catalog's three-level namespace. It's used to organize your data assets. Users can see all catalogs on which they have been assigned the USE CATALOG data permission.

Depending on how your workspace was created and enabled for Unity Catalog, your users may have default permissions on automatically provisioned catalogs, including either the main catalog or the *workspace catalog* (<workspace-name>).

Schemas

A schema (also called a database) is the second layer of Unity Catalog's three-level namespace. A schema organizes tables and views. Users can see all schemas on which they have been assigned the USE SCHEMA permission, along with the USE CATALOG permission on the schema's parent catalog. To access or list a table or view in a schema, users must also have SELECT permission on the table or view.

If your workspace was enabled for Unity Catalog manually, it includes a default schema named default in the main catalog that is accessible to all users in your workspace. If your workspace was enabled for Unity Catalog automatically and includes a <workspace-name> catalog, that catalog contains a schema named default that is accessible to all users in your workspace.

Tables

A table resides in the third layer of Unity Catalog's three-level namespace. It contains rows of data. To create a table, users must have CREATE and USE SCHEMA permissions on the schema, and they must have the USE CATALOG permission on its parent catalog. To query a table, users must have the SELECT permission on the table, the USE SCHEMA permission on its parent schema, and the USE CATALOG permission on its parent catalog.

A table can be *managed* or *external*.

Managed tables

Managed tables are the default way to create tables in Unity Catalog. Unity Catalog manages the lifecycle and file layout for these tables. You should not use tools outside of Azure Databricks to manipulate files in these tables directly. Managed tables always use the Delta table format.

For workspaces that were enabled for Unity Catalog manually, managed tables are stored in the root storage location that you configure when you create a metastore. You can optionally specify managed table storage locations at the catalog or schema levels, overriding the root storage location.

For workspaces that were enabled for Unity Catalog automatically, the metastore root storage location is optional, and managed tables are typically stored at the catalog or schema levels.

When a managed table is dropped, its underlying data is deleted from your cloud tenant within 30 days.

External tables

External tables are tables whose data lifecycle and file layout are not managed by Unity Catalog. Use external tables to register large amounts of existing data in Unity Catalog, or if you require direct access to the data using tools outside of Azure Databricks clusters or Databricks SQL warehouses.

When you drop an external table, Unity Catalog does not delete the underlying data. You can manage privileges on external tables and use them in queries in the same way as managed tables.

External tables can use the following file formats:

- DELTA
- CSV
- JSON
- AVRO
- PARQUET
- ORC
- TEXT

Views

A view is a read-only object created from one or more tables and views in a metastore. It resides in the third layer of Unity Catalog's three-level namespace. A view can be created from tables and other views in multiple schemas and catalogs. You can create dynamic views to enable row- and column-level permissions.

Volumes

A volume resides in the third layer of Unity Catalog's three-level namespace. Volumes are siblings to tables, views, and other objects organized under a schema in Unity Catalog.

Volumes contain directories and files for data stored in any format. Volumes provide non-tabular access to data, meaning that files in volumes cannot be registered as tables.

- To create a volume, users must have CREATE VOLUME and USE SCHEMA permissions on the schema, and they must have the USE CATALOG permission on its parent catalog.
- To read files and directories stored inside a volume, users must have the READ VOLUME permission, the USE SCHEMA permission on its parent schema, and the USE CATALOG permission on its parent catalog.

- To add, remove, or modify files and directories stored inside a volume, users must have WRITE VOLUME permission, the USE SCHEMA permission on its parent schema, and the USE CATALOG permission on its parent catalog.

A volume can be *managed* or *external*.

Note

When you define a volume, cloud URI access to data under the volume path is governed by the permissions of the volume.

Managed volumes

Managed volumes are a convenient solution when you want to provision a governed location for working with non-tabular files.

Managed volumes store files in the Unity Catalog default storage location for the schema in which they're contained. For workspaces that were enabled for Unity Catalog manually, managed volumes are stored in the root storage location that you configure when you create a metastore. You can optionally specify managed volume storage locations at the catalog or schema levels, overriding the root storage location. For workspaces that were enabled for Unity Catalog automatically, the metastore root storage location is optional, and managed volumes are typically stored at the catalog or schema levels.

The following precedence governs which location is used for a managed volume:

- Schema location
- Catalog location
- Unity Catalog metastore root storage location

When you delete a managed volume, the files stored in this volume are also deleted from your cloud tenant within 30 days.

External volumes

An external volume is registered to a Unity Catalog external location and provides access to existing files in cloud storage without requiring data migration. Users must have the CREATE EXTERNAL VOLUME permission on the external location to create an external volume.

External volumes support scenarios where files are produced by other systems and staged for access from within Azure Databricks using object storage or where tools outside Azure Databricks require direct file access.

Unity Catalog does not manage the lifecycle and layout of the files in external volumes. When you drop an external volume, Unity Catalog does not delete the underlying data.

Models

A model resides in the third layer of Unity Catalog's three-level namespace. In this context, "model" refers to a machine learning model that is registered in the MLflow Model Registry. To create a model in Unity Catalog, users must have the CREATE MODEL privilege for the catalog or schema. The user must also have the USE CATALOG privilege on the parent catalog and USE SCHEMA on the parent schema.

Managed storage

You can store managed tables and managed volumes at any of these levels in the Unity Catalog object hierarchy: metastore, catalog, or schema. Storage at lower levels in the hierarchy overrides storage defined at higher levels.

When an account admin creates a metastore manually, they have the option to assign a storage location in an Azure Data Lake Storage Gen2 container or Cloudflare R2 bucket in your own cloud storage account to use as metastore-level storage for managed tables and volumes. If a metastore-level managed storage location has been assigned, then managed storage locations at the catalog and schema levels are optional. That said, metastore-level storage is optional, and Databricks recommends assigning managed storage at the catalog level for logical data isolation.

Important

If your workspace was enabled for Unity Catalog automatically, the Unity Catalog metastore was created without metastore-level managed storage. You can opt to add metastore-level storage, but Databricks recommends assigning managed storage at the catalog and schema levels

Managed storage has the following properties:

- Managed tables and managed volumes store data and metadata files in managed storage.
- Managed storage locations cannot overlap with external tables or external volumes.

The following table describes how managed storage is declared and associated with Unity Catalog objects:

Expand table

Associated Unity Catalog object	How to set	Relation to external locations
Metastore	Configured by account admin during metastore creation or added after metastore creation if no storage was specified at creation.	Cannot overlap an external location.
Catalog	Specified during catalog creation using the MANAGED LOCATION keyword.	Must be contained within an external location.
Schema	Specified during schema creation using the MANAGED LOCATION keyword.	Must be contained within an external location.

The managed storage location used to store data and metadata for managed tables and managed volumes uses the following rules:

- If the containing schema has a managed location, the data is stored in the schema managed location.
- If the containing schema does not have a managed location but the catalog has a managed location, the data is stored in the catalog managed location.
- If neither the containing schema nor the containing catalog have a managed location, data is stored in the metastore managed location.

Storage credentials and external locations

To manage access to the underlying cloud storage for external tables, external volumes, and managed storage, Unity Catalog uses the following object types:

- **Storage credentials** encapsulate a long-term cloud credential that provides access to cloud storage, for example, an Azure managed identity that can access an Azure Data Lake Storage Gen2 container or a Cloudflare R2 API token.
- **External locations** contain a reference to a storage credential and a cloud storage path.

Identity management for Unity Catalog

Unity Catalog uses the identities in the Azure Databricks account to resolve users, service principals, and groups, and to enforce permissions.

To configure identities in the account, follow the instructions in Manage users, service principals, and groups. Refer to those users, service principals, and groups when you create access-control policies in Unity Catalog.

Unity Catalog users, service principals, and groups must also be added to workspaces to access Unity Catalog data in a notebook, a Databricks SQL query, Catalog Explorer, or a REST API command. The assignment of users, service principals, and groups to workspaces is called *identity federation*.

All workspaces that have a Unity Catalog metastore attached to them are enabled for identity federation.

Special considerations for groups

Any groups that already exist in the workspace are labeled **Workspace local** in the account console. These workspace-local groups cannot be used in Unity Catalog to define access policies. You must use account-level groups. If a workspace-local group is referenced in a command, that command will return an error that the group was not found. If you previously used workspace-local groups to manage access to notebooks and other artifacts, these permissions remain in effect.

Admin roles for Unity Catalog

Account admins, metastore admins, and workspace admins are involved in managing Unity Catalog:

Data permissions in Unity Catalog

In Unity Catalog, data is secure by default. Initially, users have no access to data in a metastore. Access can be granted by either a metastore admin, the owner of an object, or the owner of the catalog or schema that contains the object. Securable objects in Unity Catalog are hierarchical and privileges are inherited downward.

You can assign and revoke permissions using Catalog Explorer, SQL commands, or REST APIs.

Supported compute and cluster access modes for Unity Catalog

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 one of the Unity-Catalog-capable access modes (that is, shared or assigned), the cluster can’t access data in Unity Catalog.

Limitations for Unity Catalog vary by access mode and Databricks Runtime version.

Data lineage for Unity Catalog

You can use Unity Catalog to capture runtime data lineage across queries in any language executed on an Azure Databricks cluster or SQL warehouse. Lineage is captured down to the column level, and includes notebooks, workflows and dashboards related to the query.

Lakehouse Federation and Unity Catalog

Lakehouse Federation is the query federation platform for Azure Databricks. The term *query federation* describes a collection of features that enable users and systems to run queries against multiple siloed data sources without needing to migrate all data to a unified system.

Azure Databricks uses Unity Catalog to manage query federation. You use Unity Catalog to configure read-only *connections* to popular external database systems and create *foreign catalogs* that mirror external databases. Unity Catalog's data governance and data lineage tools ensure that data access is managed and audited for all federated queries made by the users in your Azure Databricks workspaces.

How do I set up Unity Catalog for my organization?

Follow the steps to setup Unity Catalog.

Overview of Unity Catalog enablement

To use Unity Catalog, your Azure Databricks workspaces must be enabled for Unity Catalog, which means that the workspaces are attached to a Unity Catalog metastore, the top-level container for Unity Catalog metadata.

The way admins set up Unity Catalog depends on whether the workspace was enabled automatically for Unity Catalog or requires manual enablement.

Automatic enablement of Unity Catalog

Databricks began to enable new workspaces for Unity Catalog automatically on November 9, 2023, with a rollout proceeding gradually across accounts. Workspaces that were enabled automatically have the following properties:

- An automatically-provisioned Unity Catalog metastore (unless a Unity Catalog metastore already existed for the workspace region).
- Default privileges for workspace admins, such as the ability to create a catalog or an external database connection.

- No metastore admin (unless an existing Unity Catalog metastore was used and a metastore admin was already assigned).
- No metastore-level storage for managed tables and managed volumes (unless an existing Unity Catalog metastore with metastore-level storage was used).
- A *workspace catalog*, which, when originally provisioned, is named after your workspace.

All users in your workspace can create assets in the default schema in this catalog. By default, this catalog is *bound* to your workspace, which means that it can only be accessed through your workspace. Automatic provisioning of the workspace catalog at workspace creation is rolling out gradually across accounts.

These default configurations will work well for most workspaces, but they can all be modified by a workspace admin or account admin. For example, an account admin can assign a metastore admin and create metastore-level storage, and a workspace admin can modify the workspace catalog name and access.

What if my workspace wasn’t enabled for Unity Catalog automatically?

If your workspace was not enabled for Unity Catalog automatically, an account admin or metastore admin must manually attach the workspace to a Unity Catalog metastore in the same region. If no Unity Catalog metastore exists in the region, an account admin must create one.

How do I know if my workspace was enabled for Unity Catalog?

To confirm if your workspace is enabled for Unity Catalog, ask a Azure Databricks workspace admin or account admin to check for you.

Step 1: Confirm that your workspace is enabled for Unity Catalog

In this step, you determine whether your workspace is already enabled for Unity Catalog, where enablement is defined as having a Unity Catalog metastore attached to the workspace. If your workspace is not enabled for Unity Catalog, you must enable your workspace for Unity Catalog manually.

To confirm, do one of the following.

Use the account console to confirm Unity Catalog enablement

1. As an Azure Databricks account admin, log into the account console.
2. Click **Workspaces**.
3. Find your workspace and check the **Metastore** column. If a metastore name is present, your workspace is attached to a Unity Catalog metastore and therefore enabled for Unity Catalog.

Run a SQL query to confirm Unity Catalog enablement

Run the following SQL query in the SQL query editor or a notebook that is attached to a cluster that uses *shared* or *single-user* access mode.

SQL

SELECT CURRENT_METASTORE();

If the query returns a metastore ID like the following, then your workspace is attached to a Unity Catalog metastore and therefore enabled for Unity Catalog.

Next steps if your workspace is not enabled for Unity Catalog

If your workspace is not enabled for Unity Catalog (attached to a metastore), the next step depends on whether or not you already have a Unity Catalog metastore defined for your workspace region:

- If your account already has a Unity Catalog metastore defined for your workspace region, you can simply attach your workspace to the existing metastore..
- If there is no Unity Catalog metastore defined for your workspace's region, you must create a metastore and then attach the workspace.

When your workspace is enabled for Unity Catalog, go to the next step.

Create and manage catalogs

Requirements

To create a catalog:

- You must be an Azure Databricks metastore admin or have the CREATE CATALOG privilege on the metastore.
- You must have a Unity Catalog metastore linked to the workspace where you perform the catalog creation.
- The cluster that you use to run a notebook to create a catalog must use a Unity Catalog-compliant access mode.

SQL warehouses always support Unity Catalog.

Create a catalog

To create a catalog, you can use Catalog Explorer or a SQL command.

Catalog explorer

1. Log in to a workspace that is linked to the metastore.
2. Click **Catalog**.
3. Click the **Create Catalog** button.
4. Select the catalog type that you want to create:
    - **Standard** catalog: a securable object that organizes data assets that are managed by Unity Catalog. For all use cases except Lakehouse Federation.
    - **Foreign** catalog: a securable object in Unity Catalog that mirrors a database in an external data system using Lakehouse Federation.
5. (Optional but strongly recommended) Specify a managed storage location. Requires the CREATE MANAGED STORAGE privilege on the target external location.

Important

If your workspace does not have a metastore-level storage location, you must specify a managed storage location when you create a catalog.

6. Click **Create**.
7. (Optional) Specify the workspace that the catalog is bound to.

By default, the catalog is shared with all workspaces attached to the current metastore. If the catalog will contain data that should be restricted to specific workspaces, go to the **Workspaces** tab and add those workspaces.

8. Assign permissions for your catalog.

Sql

1. Run the following SQL command in a notebook or Databricks SQL editor. Items in brackets are optional. Replace the placeholder values:
    - <catalog-name>: A name for the catalog.
    - <location-path>: Optional but strongly recommended. Provide a storage location path if you want managed tables in this catalog to be stored in a location that is different than the default root storage configured for the metastore.

Important

If your workspace does not have a metastore-level storage location, you must specify a managed storage location when you create a catalog.

This path must be defined in an external location configuration, and you must have the CREATE MANAGED STORAGE privilege on the external location configuration. You can use the path that is defined in the external location configuration or a subpath (in other words, 'abfss://my-container-name@storage-account-name.dfs.core.windows.net/finance' or 'abfss://my-container-name@storage-account-name.dfs.core.windows.net/finance/product'). Requires Databricks Runtime 11.3 and above.

- <comment>: Optional description or other comment.

Note

If you are creating a foreign catalog (a securable object in Unity Catalog that mirrors a database in an external data system, used for Lakehouse Federation), the SQL command is CREATE FOREIGN CATALOG and the options are different.

SQL

```
CREATE CATALOG [ IF NOT EXISTS ] <catalog-name>

  [ MANAGED LOCATION '<location-path>' ]

  [ COMMENT <comment> ];
```

For example, to create a catalog named example:

SQL

```
CREATE CATALOG IF NOT EXISTS example;
```

If you want to limit catalog access to specific workspaces in your account, also known as workspace-catalog binding.

For parameter descriptions.

2. Assign privileges to the catalog.

When you create a catalog, two schemas (databases) are automatically created: default and information_schema.

You can also create a catalog by using the

(Optional) Assign a catalog to specific workspaces

If you use workspaces to isolate user data access, you may want to limit catalog access to specific workspaces in your account, also known as workspace-catalog binding. The default is to share the catalog with all workspaces attached to the current metastore.

You can allow read and write access to the catalog from a workspace (the default), or you can specify read-only access. If you specify read-only, then all write operations are blocked from that workspace to that catalog.

Typical use cases for binding a catalog to specific workspaces include:

- Ensuring that users can only access production data from a production workspace environment.
- Ensuring that users can only process sensitive data from a dedicated workspace.
- Giving users read-only access to production data from a developer workspace to enable development and testing.

Workspace-catalog binding example

Take the example of production and development isolation. If you specify that your production data catalogs can only be accessed from production workspaces, this supersedes any individual grants that are issued to users.

In this diagram, prod_catalog is bound to two production workspaces. Suppose a user has been granted access to a table
in prod_catalog called my_table (using GRANT SELECT ON my_table TO <user>). If the user tries to access my_table in the Dev workspace, they receive an error message. The user can access my_table only from the Prod ETL and Prod Analytics workspaces.

Workspace-catalog bindings are respected in all areas of the platform. For example, if you query the information schema, you see only the catalogs accessible in the workspace where you issue the query. Data lineage and search UIs likewise show only the catalogs that are assigned to the workspace (whether using bindings or by default).

Bind a catalog to one or more workspaces

To assign a catalog to specific workspaces, you can use Catalog Explorer or the Unity Catalog REST API:

Permissions required: Metastore admin or catalog owner.

Note

Metastore admins can see all catalogs in a metastore using Catalog Explorer—and catalog owners can see all catalogs they own in a metastore—regardless of whether the catalog is assigned to the current workspace. Catalogs that are not assigned to the workspace appear grayed out, and no child objects are visible or queryable.

Catalog explorer

1. Log in to a workspace that is linked to the metastore.
2. Click **Catalog**.
3. In the **Catalog** pane, on the left, click the catalog name.

The main Catalog Explorer pane defaults to the **Catalogs** list. You can also select the catalog there.

4. On the **Workspaces** tab, clear the **All workspaces have access** checkbox.

If your catalog is already bound to one or more workspaces, this checkbox is already cleared.

5. Click **Assign to workspaces** and enter or find the workspaces you want to assign.
6. (Optional) Limit workspace access to read-only.

On the **Manage access level** menu, select **Change access to read-only**.

You can reverse this selection at any time by editing the catalog and selecting **Change access to read & write**.

To revoke access, go to the **Workspaces** tab, select the workspace, and click **Revoke**.

Step 2: Add users and assign the workspace admin role

The user who creates the workspace is automatically added as a workspace user with the workspace admin role (that is, a user in the admins workspace-local group). As a workspace admin, you can add and invite users to the workspace, can assign the workspace admin role to other users, and can create service principals and groups.

Account admins also have the ability to add users, service principals, and groups to your workspace. They can grant the account admin and metastore admin roles.

Manage users

Overview of user management

To manage users in Azure Databricks, you must be either an *account admin* or a *workspace admin*.

- **Account admins** can add users to the account and assign them admin roles. They can also assign users to workspaces and configure data access for them across workspaces, as long as those workspaces use identity federation.
- **Workspace admins** can add users to an Azure Databricks workspace, assign them the workspace admin role, and manage access to objects and functionality in the workspace, such as the ability to create clusters or access specified persona-based environments. Adding a user to an Azure Databricks workspace also adds them to the account.

Workspace admins are members of the admins group in the workspace, which is a reserved group that cannot be deleted.

Users with a built-in Contributor or Owner role on the workspace resource in Azure are automatically assigned the workspace admin role when they click **Launch Workspace** in the Azure portal.

Sync users to your Azure Databricks account from your Microsoft Entra ID (formerly Azure Active Directory) tenant

Account admins can sync users from your Microsoft Entra ID (formerly Azure Active Directory) tenant to your Azure Databricks account using a SCIM provisioning connector.

Manage users in your account

Account admins can add users to your Azure Databricks account using the account console. Users in a Azure Databricks account do not have any default access to a workspace, data, or compute resources.

Add users to your account using the account console

1. As an account admin, log in to the account console.
2. In the sidebar, click **User management**.
3. On the **Users** tab, click **Add User**.
4. Enter a name and email address for the user.

5. Click **Add user**.

Assign account admin roles to a user

1. As an account admin, log in to the account console.
2. In the sidebar, click **User management**.
3. Find and click the username.
4. On the **Roles** tab, turn on **Account admin** or **Marketplace admin**.

Assign a user to a workspace using the account console

To add users to a workspace using the account console, the workspace must be enabled for identity federation. Workspace admins can also assign users to workspaces using the workspace admin settings page.

1. As an account admin, log in to the account console.
2. In the sidebar, click **Workspaces**.
3. Click your workspace name.
4. On the **Permissions** tab, click **Add permissions**.
5. Search for and select the user, assign the permission level (workspace **User** or **Admin**), and click **Save**.

Remove a user from a workspace using the account console

To remove users from a workspace using the account console, the workspace must be enabled for identity federation. When a user is removed from a workspace, the user can no longer access the workspace, however permissions are maintained on the user. If the user is later added back to the workspace, they regain their previous permissions.

1. As an account admin, log in to the account console
2. In the sidebar, click **Workspaces**.
3. Click your workspace name.
4. On the **Permissions** tab, find the user.
5. Click the ⋮ kebab menu at the far right of the user row and select **Remove**.
6. On the confirmation dialog, click **Remove**.

Deactivate a user in your Azure Databricks account

Account admins can deactivate users across an Azure Databricks account. A deactivated user cannot login to the Azure Databricks account or workspaces. However, all of the user’s permissions and workspace objects remain unchanged. When a user is deactivated the following is true:

- The user cannot login to the account or any of their workspaces from any method.
- Applications or scripts that use the tokens generated by the user can no longer access the Databricks API. The tokens remain but cannot be used to authenticate while a user is deactivated.
- Notebooks owned by the user remain.
- Clusters owned by the user remain running.
- Scheduled jobs created by the user have to be assigned to a new owner to prevent them from failing.

When a user is reactivated, they can login to Azure Databricks with the same permissions. Databricks recommends deactivating users from the account instead of removing them because removing a user is a destructive action.

You cannot deactivate a user using the account console. Instead, use the Account Users API.

Remove users from your Azure Databricks account

Account admins can delete users from an Azure Databricks account. Workspace admins cannot. When you delete a user from the account, that user is also removed from their workspaces.

Important

When you remove a user from the account, that user is also removed from their workspaces, regardless of whether or not identity federation has been enabled. We recommend that you refrain from deleting account-level users unless you want them to lose access to all workspaces in the account. Be aware of the following consequences of deleting users:

- Applications or scripts that use the tokens generated by the user can no longer access Databricks APIs
- Jobs owned by the user fail
- Clusters owned by the user stop
- Queries or dashboards created by the user and shared using the Run as Owner credential have to be assigned to a new owner to prevent sharing from failing

When a user is removed from an account, the user can no longer access the account or their workspaces, however permissions are maintained on the user. If the user is later added back to the account, they regain their previous permissions.

To remove a user using the account console, do the following:

1. As an account admin, log in to the account console.
2. In the sidebar, click **User management**.
3. Find and click the username.
4. On the **User Information** tab, click the ⋮ kebab menu in the upper-right corner and select **Delete**.
5. On the confirmation dialog, click **Confirm delete**.

If you remove a user using the account console, you must ensure that you also remove the user using any SCIM provisioning connectors or SCIM API applications that have been set up for the account. If you don't, SCIM provisioning adds the user back the next time it syncs.

To remove a user from an Azure Databricks account using SCIM APIs, you must be an account admin.

Manage users in your workspace

Workspace admins can add and manage users using the workspace admin settings page.

Assign a user to a workspace using the workspace admin settings page

To add a user to a workspace using the workspace admin settings page, do the following:

1. As a workspace admin, log in to the Azure Databricks workspace.
2. Click your username in the top bar of the Azure Databricks workspace and select **Admin Settings**.
3. Click on the **Identity and access** tab.
4. Next to **Users**, click **Manage**.
5. Click **Add User**.
6. Select an existing user to assign to the workspace or click **Add new** to create a new user.

You can add any user who belongs to the Microsoft Entra ID (formerly Azure Active Directory) tenant of your Azure Databricks workspace.

7. Click **Add**.

Note

If your workspace is not enabled for **identity federation**, you only see the option to add a new user to the workspace. If you add a user that shares a username (email address) with an existing account user, those users are merged.

Assign the workspace admin role to a user using the workspace admin settings page

To assign the workspace admin role using the workspace admin settings page, do the following:

1. As a workspace admin, log in to the Azure Databricks workspace.
2. Click your username in the top bar of the Azure Databricks workspace and select **Admin Settings**.
3. Click on the **Identity and access** tab.
4. Next to **Users**, click **Manage**.
5. Select the user.
6. Click the **Entitlements** tab.
7. Click the toggle next to **Admin access**.

To remove the workspace admin role from a workspace user, perform the same steps, but clear the **Admin access** toggle.

Deactivate a user in your Azure Databricks workspace

Workspace admins can deactivate users in a Azure Databricks workspace. A deactivated user cannot login to the workspace or access it from Azure Databricks APIs, however all of the user's permissions and workspace objects remain unchanged. When a user is deactivated:

- The user cannot login to the workspaces from any method.
- The user's status shows as **Inactive** in the workspace admin setting page.
- Applications or scripts that use the tokens generated by the user can no longer access the Databricks API. The tokens remain but cannot be used to authenticate while a user is deactivated.
- Notebooks owned by the user remain.
- Clusters owned by the user remain running.
- Scheduled jobs created by the user have to be assigned to a new owner to prevent them from failing.

When a user is reactivated, they can login to the workspace with the same permissions. Databricks recommends deactivating users instead of removing them because removing a user is a destructive action. You cannot deactivate a user using the workspace admin settings page. Instead, use the Workspace Users API.

Remove a user from a workspace using the workspace admin settings page

When a user is removed from a workspace, the user can no longer access the workspace, however permissions are maintained on the user. If the user is later added back to the workspace, they regain their previous permissions.

1. As a workspace admin, log in to the Azure Databricks workspace.
2. Click your username in the top bar of the Azure Databricks workspace and select **Admin Settings**.
3. Click on the **Identity and access** tab.
4. Next to **Users**, click **Manage**.
5. Find the user and ⋮ kebab menu at the far right of the user row and select **Remove**.
6. Click **Delete** to confirm.

(Recommended) Sync account-level identities from Microsoft Entra ID (formerly Azure Active Directory)

It can be convenient to manage user access to Azure Databricks by setting up provisioning from Microsoft Entra ID (formerly Azure Active Directory). For complete instructions,

Step 3: Create clusters or SQL warehouses that users can use to run queries and create objects

To run Unity Catalog workloads, compute resources must comply with certain security requirements. Non-compliant compute resources cannot access data or other objects in Unity Catalog. SQL warehouses always comply with Unity Catalog requirements, but some cluster access modes do not.

As a workspace admin, you can opt to make compute creation restricted to admins or let users create their own SQL warehouses and clusters. You can also create cluster policies that enable users to create their own clusters, using Unity Catalog-compliant specifications that you enforce.

Step 4: Grant privileges to users

To create objects and access them in Unity Catalog catalogs and schemas, a user must have permission to do so. This section describes the user and admin privileges granted on some workspaces by default and describes how to grant additional privileges.

Default user privileges

Some workspaces have default user (non-admin) privileges upon launch:

- If your workspace launched with an automatically-provisioned workspace catalog, all workspace users can create objects in the workspace catalog's default schema.
- If your workspace was enabled for Unity Catalog manually, it has a main catalog provisioned automatically.

Workspace users have the USE CATALOG privilege on the main catalog, which doesn't grant the ability to create or select from any objects in the catalog, but is a prerequisite for working with any objects in the catalog. The user who created the metastore owns the main catalog by default and can both transfer ownership and grant access to other users.

If metastore storage is added after the metastore is created, no main catalog is provisioned.

Other workspaces have no catalogs created by default and no non-admin user privileges enabled by default. A workspace admin must create the first catalog and grant users access to it and the objects in it. Skip ahead to Step 5: Create new catalogs and schemas before you complete the steps in this section.

Default admin privileges

Some workspaces have default workspace admin privileges upon launch:

- If your workspace was enabled for Unity Catalog automatically:
    - Workspace admins can create new catalogs and objects in new catalogs, and grant access to them.
    - There is no metastore admin by default.
    - Workspace admins own the workspace catalog (if there is one) and can grant access to that catalog and any objects in that catalog.
- If your workspace was enabled for Unity Catalog manually:
    - Workspace admins have no special Unity Catalog privileges by default.
    - Metastore admins must exist and can create any Unity Catalog object and can take ownership of any Unity Catalog object.

For a list of additional object privileges granted to workspace admins in automatically-enabled Unity Catalog workspaces.

Grant privileges

For access to objects other than those listed in the previous sections, a privileged user must grant that access.

For example, to grant a group the ability to create new schemas in my-catalog, the catalog owner can run the following in the SQL Editor or a notebook:

SQL

GRANT CREATE SCHEMA ON my-catalog TO `data-consumers`;

If your workspace was enabled for Unity Catalog automatically, the workspace admin owns the workspace catalog and can grant the ability to create new schemas:

SQL

GRANT CREATE SCHEMA ON <workspace-catalog> TO `data-consumers`;

You can also grant and revoke privileges using Catalog Explorer.

Important

You cannot grant privileges to the workspace-local users or admins groups. To grant privileges on groups, they must be account-level groups.

Step 5: Create new catalogs and schemas

To start using Unity Catalog, you must have at least one catalog defined. Catalogs are the primary unit of data isolation and organization in Unity Catalog. All schemas and tables live in catalogs, as do volumes, views, and models.

Some workspaces have no automatically-provisioned catalog. To use Unity Catalog, a workspace admin must create the first catalog for such workspaces.

Other workspaces have access to a pre-provisioned catalog that your users can access to get started (either the workspace catalog or the main catalog, depending on how your workspace was enabled for Unity Catalog). As you add more data and AI assets into Azure Databricks, you can create additional catalogs to group those assets in a way that makes it easy to govern data logically.

As a metastore admin, workspace admin (auto-enabled workspaces only), or other user with the CREATE CATALOG privilege, you can create new catalogs in the metastore. When you do, you should:

1. **Create *managed storage* for the new catalog.**

Managed storage is a dedicated storage location in your Azure account for managed tables and managed volumes. You can assign managed storage to the metastore, to catalogs, and to schemas. When a user creates a table, the data is stored in the storage location that is lowest in the hierarchy. For example, if a storage location is defined for the metastore and catalog but not the schema, the data is stored in the location defined for the catalog.

Databricks recommends that you assign managed storage at the catalog level, because catalogs typically represent logical units of data isolation. If you are comfortable with data in multiple catalogs sharing the same storage location, you can default to the metastore-level storage location. If your workspace was enabled for Unity Catalog automatically, there is no metastore-level storage by default. An account admin has the option to configure metastore-level storage.

Assigning managed storage to a catalog requires that you create:

- A *storage credential*.
- An *external location* that references that storage credential.

Connect to cloud object storage using Unity Catalog

Databricks recommends using Unity Catalog to manage access to all data stored in cloud object storage. Unity Catalog provides a suite of tools to configure secure connections to cloud object storage. These connections provide access to complete the following actions:

- Ingest raw data into a lakehouse.
- Create and read managed tables in secure cloud storage.
- Register or create external tables containing tabular data.
- Read and write unstructured data.

Warning

Do not give end users storage-level access to Unity Catalog managed tables or volumes. This compromises data security and governance.

Granting users direct storage-level access to external location storage in Azure Data Lake Storage Gen2 does not honor any permissions granted or audits maintained by Unity Catalog. Direct access will bypass auditing, lineage, and other security and monitoring features of Unity Catalog, including access control and permissions. You are responsible for managing direct storage access through Azure Data Lake Storage Gen2 and ensuring that users have the appropriate permissions granted via Fabric.

Avoid all scenarios that grant direct storage-level write access for buckets that store Databricks managed tables. Modifying, deleting, or evolving any objects directly through storage that were originally managed by Unity Catalog can result in data corruption.

How does Unity Catalog connect object storage to Azure Databricks?

Azure Databricks supports both Azure Data Lake Storage Gen2 containers and Cloudflare R2 buckets (Public Preview) as cloud storage locations for data and AI assets registered in Unity Catalog. R2 is intended primarily for uses cases in which you want to avoid data egress fees, such as Delta Sharing across clouds and regions.

To manage access to the underlying cloud storage that holds tables and volumes, Unity Catalog uses the following object types:

- A *storage credential* represents an authentication and authorization mechanism for accessing data stored on your cloud tenant, using an Azure managed identity for Azure Data Lake Storage Gen2 containers or an R2 API token for Cloudflare R2 buckets. Each storage credential is subject to Unity Catalog access-control policies that control which users and groups can access the credential. If a user does not have access to a storage credential in Unity Catalog, the request fails and Unity Catalog does not attempt to authenticate to your cloud tenant on the user's behalf. Permission to create storage credentials should only be granted to users who need to define external locations.
- An *external location* is an object that combines a cloud storage path with a storage credential that authorizes access to the cloud storage path. Each storage location is subject to Unity Catalog access-control policies that control which users and groups can access the credential. If a user does not have access to a storage location in Unity Catalog, the request fails and Unity Catalog does not attempt to authenticate to your cloud tenant on the user's behalf. Permission to create and use external locations should only be granted to users who need to create external tables, external volumes, or managed storage locations.

External locations are used both for external data assets, like *external tables* and *external volumes*, and for *managed* data assets, like *managed tables* and *managed volumes*.

When an external location is used for storing *managed tables* and *managed volumes*, it is called a *managed storage location*. Managed storage locations can exist at the metastore, catalog, or schema level. Databricks recommends configuring managed storage locations at the catalog level. If you need more granular isolation, you can specify managed storage locations at the schema level. Workspaces that are enabled for Unity Catalog automatically have no metastore-level storage by default, but you can specify a managed storage location at the metastore level to provide default storage when no catalog-level storage is defined. Workspaces that are enabled for Unity Catalog manually receive a metastore-level managed storage location by default.

Volumes are the securable object that most Azure Databricks users should use to interact directly with non-tabular data in cloud object storage.

2. **Bind the new catalog to your workspace if you want to limit access from other workspaces that share the same metastore.**
3. **Grant privileges on the catalog.**

Catalog creation example

The following example shows the creation of a catalog with managed storage, followed by granting the SELECT privilege on the catalog:

SQL

CREATE CATALOG IF NOT EXISTS mycatalog

MANAGED LOCATION 'abfss://mycontainer@<myaccount.dfs.core.windows.net//depts/finance';

GRANT SELECT ON mycatalog TO `finance-team`;

Create a schema

Schemas represent more granular groupings (like departments or projects, for example) than catalogs. All tables and other Unity Catalog objects in the catalog are contained in schemas. As the owner of a new catalog, you may want to create the schemas in the catalog. But you might want instead to delegate the ability to create schemas to other users, by giving them the CREATE SCHEMA privilege on the catalog.

Create and manage schemas (databases)

Requirements

- You must have a Unity Catalog metastore linked to the workspace where you perform the schema creation.
- You must have the USE CATALOG and CREATE SCHEMA data permissions on the schema’s parent catalog. Either a metastore admin or the owner of the catalog can grant you these privileges. If you are a metastore admin, you can grant these privileges to yourself.
- The cluster that you use to run a notebook to create a schema must use a Unity Catalog-compliant access mode. See Access modes.

SQL warehouses always support Unity Catalog.

Create a schema

To create a schema, you can use Catalog Explorer or SQL commands.

Catalog explorer

1. Log in to a workspace that is linked to the metastore.
2. Click **Catalog**.
3. In the **Catalog** pane on the left, click the catalog you want to create the schema in.
4. In the detail pane, click **Create schema**.
5. Give the schema a name and add any comment that would help users understand the purpose of the schema.
6. (Optional) Specify a managed storage location. Requires the CREATE MANAGED STORAGE privilege on the target external location.
7. Click **Create**.
8. Assign permissions for your catalog.
9. Click **Save**.

Sql

1. Run the following SQL commands in a notebook or Databricks SQL editor. Items in brackets are optional. You can use either SCHEMA or DATABASE. Replace the placeholder values:
    - <catalog-name>: The name of the parent catalog for the schema.
    - <schema-name>: A name for the schema.
    - <location-path>: Optional. Requires additional privileges.
    - <comment>: Optional description or other comment.
    - <property-key> = <property-value> [, ...]: Optional. Spark SQL properties and values to set for the schema.

SQL

```
USE CATALOG <catalog>;

CREATE { DATABASE | SCHEMA } [ IF NOT EXISTS ] <schema-name>

   [ MANAGED LOCATION '<location-path>' ]

   [ COMMENT <comment> ]

   [ WITH DBPROPERTIES ( <property-key = property_value [ , ... ]> ) ];
```

You can optionally omit the USE CATALOG statement and replace <schema-name> with <catalog-name>.<schema-name>.

2. Assign privileges to the schema.

Delete a schema

To delete (or drop) a schema, you can use Catalog Explorer or a SQL command. To drop a schema you must be its owner.

Catalog explorer

You must delete all tables in the schema before you can delete it.

1. Log in to a workspace that is linked to the metastore.
2. Click **Catalog**.
3. In the **Catalog** pane, on the left, click the schema that you want to delete.
4. In the detail pane, click the three-dot menu in the upper right corner and select **Delete**.
5. On the **Delete schema** dialog, click **Delete**.

Sql

Run the following SQL command in a notebook or Databricks SQL editor. Items in brackets are optional. Replace the placeholder <schema-name>.

If you use DROP SCHEMA without the CASCADE option, you must delete all tables in the schema before you can delete it.

SQL

```
DROP SCHEMA [ IF EXISTS ] <schema-name> [ RESTRICT | CASCADE ]
```

For example, to delete a schema named inventory_schema and its tables:

SQL

```
DROP SCHEMA inventory_schema CASCADE
```

(Optional) Assign the metastore admin role

If your workspace was enabled for Unity Catalog automatically, no metastore admin role is assigned by default. Metastore admins have some privileges that workspace admins don’t.

You might want to assign a metastore admin if you need to:

- Change ownership of catalogs after someone leaves the company.
- Manage and delegate permissions on the init script and jar allowlist.
- Delegate the ability to create catalogs and other top-level permissions to non-workspace admins.
- Receive shared data through Delta Sharing.
- Remove default workspace admin permissions.
- Add managed storage to the metastore, if it has none.

(Optional) Keep working with your Hive metastore

If your workspace was in service before it was enabled for Unity Catalog, it likely has a Hive metastore that contains data that you want to continue to use. Databricks recommends that you migrate the tables managed by the Hive metastore to the Unity Catalog metastore, but if you choose not to, you can continue to work with the Hive metastore.

The Hive metastore is represented in Unity Catalog interfaces as a catalog named hive_metastore. In order to continue working with data in your Hive metastore without having to update queries to specify the hive_metastore catalog, you can set the workspace's default catalog to hive_metastore.

Depending on when your workspace was enabled for Unity Catalog, the default catalog may already be hive_metastore.

(Optional) Create metastore-level storage

Although Databricks recommends that you create a separate managed storage location for each catalog in your metastore (and you can do the same for schemas), you can opt instead to create a managed location at the metastore level and use it as the default storage for multiple catalogs and schemas.

If you want metastore-level storage, you must also assign a metastore admin.

Metastore-level storage is **required** only if the following are true:

- You want to share notebooks using Databricks-to-Databricks Delta Sharing.
- You use a Databricks partner product integration that relies on personal staging locations (deprecated).

How do I know if my workspace includes a *workspace catalog* ?

Some new workspaces have a *workspace catalog*, which, when originally provisioned, is named after your workspace. To determine if your workspace has one, click **Catalog** in the sidebar to open Catalog Explorer, and search for a catalog that uses your workspace name as the catalog name.

Supported data file formats

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.

Unity Catalog limitations

Unity Catalog has the following limitations.

Note

If your cluster is running on a Databricks Runtime version below 11.3 LTS, there may be additional limitations, not listed here. Unity Catalog is supported on Databricks Runtime 11.3 LTS or above.

Unity Catalog limitations vary by Databricks Runtime and access mode. Structured Streaming workloads have additional limitations based on Databricks Runtime and access mode.

- Workloads in R do not support the use of dynamic views for row-level or column-level security.
- In Databricks Runtime 13.1 and above, shallow clones are supported to create Unity Catalog managed tables from existing Unity Catalog managed tables. In Databricks Runtime 13.0 and below, there is no support for shallow clones in Unity Catalog.
- 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.
- Custom partition schemes created using commands like ALTER TABLE ADD PARTITION are not supported for tables in Unity Catalog. Unity Catalog can access tables that use directory-style partitioning.
- Overwrite mode for DataFrame write operations into Unity Catalog is supported only for Delta tables, not for other file formats. The user must have the CREATE privilege on the parent schema and must be the owner of the existing object or have the MODIFY privilege on the object.
- In Databricks Runtime 13.2 and above, Python scalar UDFs are supported. In Databricks Runtime 13.1 and below, you cannot use Python UDFs, including UDAFs, UDTFs, and Pandas on Spark (applyInPandas and mapInPandas).

- In Databricks Runtime 14.2 and above, Scala scalar UDFs are supported on shared clusters. In Databricks Runtime 14.1 and below, all Scala UDFs are not supported on shared clusters.
- 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 (formerly Azure Active Directory) connectors, and Terraform) to reference account endpoints instead of workspace endpoints.
- 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.
- Audit logging is supported for Unity Catalog events at the workspace level only. Events that take place at the account level without reference to a workspace, such as creating a metastore, are not logged.

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.

Resource quotas

Unity Catalog enforces resource quotas on all securable objects. Limits respect the same hierarchical organization throughout Unity Catalog. If you expect to exceed these resource limits, contact your Azure Databricks account team.

Quota values below are expressed relative to the parent (or grandparent) object in Unity Catalog.

Expand table

Object	**Parent**	**Value**
table	schema	10000
table	metastore	100000
volume	schema	10000
function	schema	10000
registered model	schema	1000
registered model	metastore	5000
model version	registered model	10000
model version	metastore	100000
schema	catalog	10000
catalog	metastore	1000
connection	metastore	1000
storage credential	metastore	200
external location	metastore	500

Create tables in Unity Catalog

Managed tables

Managed tables are the default way to create tables in Unity Catalog. Unity Catalog manages the lifecycle and file layout for these tables. You should not use tools outside of Azure Databricks to manipulate files in these tables directly.

Managed tables are stored in *managed storage*, either at the metastore, catalog, or schema level, depending on how the schema and catalog are configured.

Managed tables always use the Delta table format.

When a managed table is dropped, its underlying data is deleted from your cloud tenant within 30 days.

External tables

External tables are tables whose data is stored outside of the managed storage location specified for the metastore, catalog, or schema. Use external tables only when you require direct access to the data outside of Azure Databricks clusters or Databricks SQL warehouses.

When you run DROP TABLE on an external table, Unity Catalog does not delete the underlying data. To drop a table you must be its owner. You can manage privileges on external tables and use them in queries in the same way as managed tables. To create an external table with SQL, specify a LOCATION path in your CREATE TABLE statement. External tables can use the following file formats:

- DELTA
- CSV
- JSON
- AVRO
- PARQUET
- ORC
- TEXT

To manage access to the underlying cloud storage for an external table, you must set up storage credentials and external locations.

Requirements

You must have the CREATE TABLE privilege on the schema in which you want to create the table, as well as the USE SCHEMA privilege on the schema and the USE CATALOG privilege on the parent catalog.

If you are creating an external table.

Create a managed table

To create a managed table, run the following SQL command. You can also use the example notebook shared by Github URL: unity-catalog-external-table-example-azure.dbc (github.com) to create a table. Items in brackets are optional. Replace the placeholder values:

- <catalog-name>: The name of the catalog that will contain the table..

This cannot be the hive_metastore catalog that is created automatically for the Hive metastore associated with your Azure Databricks workspace. You can drop the catalog name if you are creating the table in the workspace's default catalog.

- <schema-name>: The name of the schema that will contain the table..
- <table-name>: A name for the table.
- <column-specification>: The name and data type for each column.

SQL

```
CREATE TABLE <catalog-name>.<schema-name>.<table-name>

(

 <column-specification>

);
```

Python

```
spark.sql("CREATE TABLE <catalog-name>.<schema-name>.<table-name> "

 "("

 "  <column-specification>"

 ")")
```

R

```
library(SparkR)
```

```
sql(paste("CREATE TABLE <catalog-name>.<schema-name>.<table-name> ",
 "(",
 "  <column-specification>",
 ")",
 sep = ""))
```

Scala

```
spark.sql("CREATE TABLE <catalog-name>.<schema-name>.<table-name> " +
 "(" +
 "  <column-specification>" +
 ")")
```

For example, to create the table main.default.department and insert five rows into it:

SQL

```
CREATE TABLE main.default.department
(
 deptcode  INT,
 deptname  STRING,
 location  STRING
);

INSERT INTO main.default.department VALUES
 (10, 'FINANCE', 'EDINBURGH'),
 (20, 'SOFTWARE', 'PADDINGTON'),
 (30, 'SALES', 'MAIDSTONE'),
 (40, 'MARKETING', 'DARLINGTON'),
 (50, 'ADMIN', 'BIRMINGHAM');
```

Python

```
spark.sql("CREATE TABLE main.default.department "
 "("
 "  deptcode  INT,"
 "  deptname  STRING,"
 "  location  STRING"
 ")"
 "INSERT INTO main.default.department VALUES "
 "  (10, 'FINANCE', 'EDINBURGH'),"
 "  (20, 'SOFTWARE', 'PADDINGTON'),"
 "  (30, 'SALES', 'MAIDSTONE'),"
 "  (40, 'MARKETING', 'DARLINGTON'),"
 "  (50, 'ADMIN', 'BIRMINGHAM')")
```

R

```
library(SparkR)

sql(paste("CREATE TABLE main.default.department ",
 "(",
 "  deptcode  INT,",
 "  deptname  STRING,",
 "  location  STRING",
 ")",
 "INSERT INTO main.default.department VALUES ",
 "  (10, 'FINANCE', 'EDINBURGH'),",
 "  (20, 'SOFTWARE', 'PADDINGTON'),",
 "  (30, 'SALES', 'MAIDSTONE'),",
 "  (40, 'MARKETING', 'DARLINGTON'),",
 "  (50, 'ADMIN', 'BIRMINGHAM')",
 sep = ""))
```

Scala

```
spark.sql("CREATE TABLE main.default.department " +
"(" +
"  deptcode  INT," +
"  deptname  STRING," +
"  location  STRING" +
")" +
"INSERT INTO main.default.department VALUES " +
"  (10, 'FINANCE', 'EDINBURGH')," +
"  (20, 'SOFTWARE', 'PADDINGTON')," +
"  (30, 'SALES', 'MAIDSTONE')," +
"  (40, 'MARKETING', 'DARLINGTON')," +
"  (50, 'ADMIN', 'BIRMINGHAM')")
```

Example notebook: Create managed tables

You can use the following example notebooks to create a catalog, schema, and managed table, and to manage permissions on them.

Create and manage a table in Unity Catalog using Python notebook

unity-catalog-quickstart-python.dbc (github.com)

Create and manage a table in Unity Catalog using SQL notebook

unity-catalog-example-notebook.dbc (github.com)

Drop a managed table

You must be the table's owner to drop a table. To drop a managed table, run the following SQL command:

SQL

DROP TABLE IF EXISTS catalog_name.schema_name.table_name;

When a managed table is dropped, its underlying data is deleted from your cloud tenant within 30 days.

Create an external table

The data in an external table is stored in a path on your cloud tenant. To work with external tables, Unity Catalog introduces two objects to access and work with external cloud storage:

- A *storage credential* contains an authentication method for accessing a cloud storage location. The storage credential does not contain a mapping to the path to which it grants access. Storage credentials are access-controlled to determine which users can use the credential.
- An *external location* maps a storage credential with a cloud storage path to which it grants access. The external location grants access only to that cloud storage path and its contents. External locations are access-controlled to determine which users can use them. An external location is used automatically when your SQL command contains a LOCATION clause.

Requirements

To create an external table, you must have:

- The CREATE EXTERNAL TABLE privilege on an external location that grants access to the LOCATION accessed by the external table.
- The USE SCHEMA permission on the table's parent schema.
- The USE CATALOG permission on the table's parent catalog.
- The CREATE TABLE permission on the table's parent schema.

External locations and storage credentials are stored at the metastore level, rather than in a catalog. To create a storage credential, you must be an account admin or have the CREATE STORAGE CREDENTIAL privilege. To create an external location, you must be the metastore admin or have the CREATE EXTERNAL LOCATION privilege.

Create a table

Use one of the following command examples in a notebook or the SQL query editor to create an external table.

You can also use an unity-catalog-external-table-example-azure.dbc (github.com) to create the storage credential, external location, and external table, and also manage permissions for them.

In the following examples, replace the placeholder values:

- <catalog>: The name of the catalog that will contain the table.

This cannot be the hive_metastore catalog that is created automatically for the Hive metastore associated with your Azure Databricks workspace. You can drop the catalog name if you are creating the table in the workspace's default catalog.

- <schema>: The name of the schema that will contain the table.
- <table-name>: A name for the table.

- <column-specification>: The name and data type for each column.
- <bucket-path>: The path to the cloud storage bucket where the table will be created.
- <table-directory>: A directory where the table will be created. Use a unique directory for each table.

Important

Once a table is created in a path, users can no longer directly access the files in that path from Azure Databricks even if they have been given privileges on an external location or storage credential to do so. This is to ensure that users cannot circumvent access controls applied to tables by reading files from your cloud tenant directly.

SQL

```
CREATE TABLE <catalog>.<schema>.<table-name>
(
 <column-specification>
)
LOCATION 'abfss://<bucket-path>/<table-directory>';
```

Python

```
spark.sql("CREATE TABLE <catalog>.<schema>.<table-name> "
 "("
 "  <column-specification>"
 ") "
 "LOCATION 'abfss://<bucket-path>/<table-directory>'")
```

R

```
library(SparkR)

sql(paste("CREATE TABLE <catalog>.<schema>.<table-name> ",
 "(",
 "  <column-specification>",
 ") ",
 "LOCATION 'abfss://<bucket-path>/<table-directory>'",
 sep = ""))
```

Scala

```
spark.sql("CREATE TABLE <catalog>.<schema>.<table-name> " +
 "(" +
 "  <column-specification>" +
 ") " +
 "LOCATION 'abfss://<bucket-path>/<table-directory>'")
```

Unity Catalog checks that you have the following permissions:

- CREATE EXTERNAL TABLE on the external location that references the cloud storage path you specify.
- CREATE TABLE on the parent schema.
- USE SCHEMA on the parent schema.
- USE CATALOG on the parent catalog.

If you do, the external table is created. Otherwise, an error occurs and the external table is not created.

Note

You can instead migrate an existing external table in the Hive metastore to Unity Catalog without duplicating its data.

Example notebook: Create external tables

You can use the following example notebook to create a catalog, schema, and external table, and to manage permissions on them.

Create and manage an external table in Unity Catalog notebook

unity-catalog-external-table-example-azure.dbc (github.com)

Create a table from files stored in your cloud tenant

You can populate a managed or external table with records from files stored in your cloud tenant. Unity Catalog reads the files at that location and inserts their contents into the table. In Unity Catalog, this is called *path-based-access*.

Explore the contents of the files

To explore data stored in an external location before you create tables from that data, you can use Catalog Explorer or the following commands.

Permissions required: You must have the READ FILES permission on the external location associated with the cloud storage path to return a list of data files in that location.

Sql

1. List the files in a cloud storage path:

SQL

```
LIST 'abfss://<path-to-files>';
```

2. Query the data in the files in a given path:

SQL

```
SELECT * FROM <format>.`abfss://<path-to-files>`;
```

Python

1. List the files in a cloud storage path:

Python

```
display(spark.sql("LIST 'abfss://<path-to-files>'"))
```

2. Query the data in the files in a given path:

Python

```
display(spark.read.load("abfss://<path-to-files>"))
```

R

1. List the files in a cloud storage path:

R

```
library(SparkR)


display(sql("LIST 'abfss://<path-to-files>'"))
```

2. Query the data in the files in a given path:

R

```
library(SparkR)
```

```
display(loadDF("abfss://<path-to-files>"))
```

Scala

1. List the files in a cloud storage path:

Scala

```
display(spark.sql("LIST 'abfss://<path-to-files>'"))
```

2. Query the data in the files in a given path:

Scala

```
display(spark.read.load("abfss://<path-to-files>"))
```

Create a table from the files

Follow the examples in this section to create a new table and populate it with data files on your cloud tenant.

Note

You can instead migrate an existing external table in the Hive metastore to Unity Catalog without duplicating its data.

Important

- When you create a table using this method, the storage path is read only once, to prevent duplication of records. If you want to re-read the contents of the directory, you must drop and re-create the table
- The bucket path where you create a table cannot also be used to read or write data files.
- Only the files in the exact directory are read; the read is not recursive.
- You must have the following permissions:
    - USE CATALOG on the parent catalog and USE SCHEMA on the schema.
    - CREATE TABLE on the parent schema.
    - READ FILES on the external location associated with the bucket path where the files are located, or directly on the storage credential if you are not using an external location.
    - If you are creating an external table, you need CREATE EXTERNAL TABLE on the bucket path where the table will be created.

To create a new managed table and populate it with data in your cloud storage, use the following examples.

SQL

```
CREATE TABLE <catalog>.<schema>.<table-name>
(
  <column-specification>
)
SELECT * from <format>.`abfss://<path-to-files>`;
```

Python

```
spark.sql("CREATE TABLE <catalog>.<schema>.<table-name> "
  "( "
  "  <column-specification> "
  ") "
  "SELECT * from <format>.`abfss://<path-to-files>`")
```

R

```
library(SparkR)

sql(paste("CREATE TABLE <catalog>.<schema>.<table-name> ",
  "( ",
  "  <column-specification> ",
  ") ",
  "SELECT * from <format>.`abfss://<path-to-files>`",
  sep = ""))
```

Scala

```
spark.sql("CREATE TABLE <catalog>.<schema>.<table-name> " +
  "( " +
  "  <column-specification> " +
  ") " +
  "SELECT * from <format>.`abfss://<path-to-files>`")
```

To create an external table and populate it with data in your cloud storage, add a LOCATION clause:

SQL

```
CREATE TABLE <catalog>.<schema>.<table-name>
(
  <column-specification>
)
USING <format>
LOCATION 'abfss://<table-location>'
SELECT * from <format>.`abfss://<path-to-files>`;
```

Python

```
spark.sql("CREATE TABLE <catalog>.<schema>.<table-name> "
  "( "
  "  <column-specification> "
  ") "
  "USING <format> "
  "LOCATION 'abfss://<table-location>' "
  "SELECT * from <format>.`abfss://<path-to-files>`")
```

R

```
library(SparkR)

sql(paste("CREATE TABLE <catalog>.<schema>.<table-name> ",
  "( ",
  "  <column-specification> ",
  ") ",
  "USING <format> ",
  "LOCATION 'abfss://<table-location>' ",
  "SELECT * from <format>.`abfss://<path-to-files>` ",
  sep = ""))
```

Scala

```
spark.sql("CREATE TABLE <catalog>.<schema>.<table-name> " +
 "( " +
 "  <column-specification> " +
 ") " +
 "USING <format> " +
 "LOCATION 'abfss://<table-location>' " +
 "SELECT * from <format>.`abfss://<path-to-files>` ")
```

Insert records from a path into an existing table

To insert records from a bucket path into an existing table, use the COPY INTO command. In the following examples, replace the placeholder values:

- <catalog>: The name of the table's parent catalog.
- <schema>: The name of the table's parent schema.
- <path-to-files>: The bucket path that contains the data files.
- <format>: The format of the files, for example delta.
- <table-location>: The bucket path where the table will be created.

Important

- When you insert records into a table using this method, the bucket path you provide is read only once, to prevent duplication of records.
- The bucket path where you create a table cannot also be used to read or write data files.
- Only the files in the exact directory are read; the read is not recursive.
- You must have the following permissions:
  - USE CATALOG on the parent catalog and USE SCHEMA on the schema.
  - MODIFY on the table.

- READ FILES on the external location associated with the bucket path where the files are located, or directly on the storage credential if you are not using an external location.
- To insert records into an external table, you need CREATE EXTERNAL TABLE on the bucket path where the table is located.

To insert records from files in a bucket path into a managed table, using an external location to read from the bucket path:

SQL

```
COPY INTO <catalog>.<schema>.<table>
FROM (
 SELECT *
 FROM 'abfss://<path-to-files>'
)
FILEFORMAT = <format>;
```

Python

```
spark.sql("COPY INTO <catalog>.<schema>.<table> "
 "FROM ( "
 "  SELECT * "
 "  FROM 'abfss://<path-to-files>' "
 ") "
 "FILEFORMAT = <format>")
```

R

```
library(SparkR)

sql(paste("COPY INTO <catalog>.<schema>.<table> ",
 "FROM ( ",
 "  SELECT * ",
 "  FROM 'abfss://<path-to-files>' ",
 ") ",
 "FILEFORMAT = <format>",
 sep = ""))
```

Scala

```
spark.sql("COPY INTO <catalog>.<schema>.<table> " +
 "FROM ( " +
 "  SELECT * " +
 "  FROM 'abfss://<path-to-files>' " +
 ") " +
 "FILEFORMAT = <format>")
```

To insert into an external table, add a LOCATION clause:

SQL

```
COPY INTO <catalog>.<schema>.<table>
LOCATION 'abfss://<table-location>'
FROM (
 SELECT *
 FROM 'abfss://<path-to-files>'
)
FILEFORMAT = <format>;
```

Python

```
spark.sql("COPY INTO <catalog>.<schema>.<table> "
 "LOCATION 'abfss://<table-location>' "
 "FROM ( "
 "  SELECT * "
 "  FROM 'abfss://<path-to-files>' "
 ") "
 "FILEFORMAT = <format>")
```

R

```
library(SparkR)
```

```
sql(paste("COPY INTO <catalog>.<schema>.<table> ",
 "LOCATION 'abfss://<table-location>' ",
 "FROM ( ",
 "  SELECT * ",
 "  FROM 'abfss://<path-to-files>' ",
 ") ",
 "FILEFORMAT = <format>",
 sep = ""))
```

Scala

```
spark.sql("COPY INTO <catalog>.<schema>.<table> " +
 "LOCATION 'abfss://<table-location>' " +
 "FROM ( " +
 "  SELECT * " +
 "  FROM 'abfss://<path-to-files>' " +
 ") " +
 "FILEFORMAT = <format>")
```

Add comments to a table

As a table owner or a user with the MODIFY privilege on a table, you can add comments to a table and its columns. You can add comments using the following functionality:

- The COMMENT ON command. This option does not support column comments.
- The COMMENT option when you use the CREATE TABLE and ALTER TABLE commands. This option supports column comments.
- The “manual” comment field in Catalog Explorer. This option supports column comments.
- AI-generated comments (also known as AI-generated documentation) in Catalog Explorer. You can view a comment suggested by a large language model (LLM) that takes into account the table metadata, such as the table schema and column names, and edit or accept the comment as-is to add it.

Create views

A view is a read-only object composed from one or more tables and views in a metastore. It resides in the third layer of Unity Catalog's three-level namespace. A view can be created from tables and other views in multiple schemas and catalogs.

Dynamic views can be used to provide row- and column-level access control, in addition to data masking.

Example syntax for creating a view:

SQL

```
CREATE VIEW main.default.experienced_employee

 (id COMMENT 'Unique identification number', Name)

 COMMENT 'View for experienced employees'

AS SELECT id, name

  FROM all_employee

  WHERE working_years > 5;
```

Note

Views might have different execution semantics if they're backed by data sources other than Delta tables. Databricks recommends that you always define views by referencing data sources using a table or view name. Defining views against datasets by specifying a path or URI can lead to confusing data governance requirements.

Requirements

To create a view:

- You must have the USE CATALOG permission on the parent catalog and the USE SCHEMA and CREATE TABLE permissions on the parent schema. A metastore admin or the catalog owner can grant you all of these privileges. A schema owner can grant you USE SCHEMA and CREATE TABLE privileges on the schema.
- You must be able to read the tables and views referenced in the view (SELECT on the table or view, as well as USE CATALOG on the catalog and USE SCHEMA on the schema).
- If a view references tables in the workspace-local Hive metastore, the view can be accessed only from the workspace that contains the workspace-local tables. For this reason, Databricks recommends creating views only from tables or views that are in the Unity Catalog metastore.

- You cannot create a view that references a view that has been shared with you using Delta Sharing.

To read a view, the permissions required depend on the compute type and access mode:

- For shared clusters and SQL warehouses, you need SELECT on the view itself, USE CATALOG on its parent catalog, and USE SCHEMA on its parent schema.
- For single-user clusters, you must also have SELECT on all tables and views that the view references, in addition to USE CATALOG on their parent catalogs and USE SCHEMA on their parent schemas.

To create or read dynamic views:

- Requirements for dynamic views are the same as those listed in the preceding sections, except that you must use a shared cluster or SQL warehouse to create or read a dynamic view. You cannot use single-user clusters.

Create a view

To create a view, run the following SQL command. Items in brackets are optional. Replace the placeholder values:

- <catalog-name>: The name of the catalog.
- <schema-name>: The name of the schema.
- <view-name>: A name for the view.
- <query>: The query, columns, and tables and views used to compose the view.

SQL

```
CREATE VIEW <catalog-name>.<schema-name>.<view-name> AS

SELECT <query>;
```

Python

```
spark.sql("CREATE VIEW <catalog-name>.<schema-name>.<view-name> AS "

  "SELECT <query>")
```

R

```
library(SparkR)


sql(paste("CREATE VIEW <catalog-name>.<schema-name>.<view-name> AS ",
  "SELECT <query>",
  sep = ""))
```

Scala

```
spark.sql("CREATE VIEW <catalog-name>.<schema-name>.<view-name> AS " +
  "SELECT <query>")
```

For example, to create a view named sales_redacted from columns in the sales_raw table:

SQL

```
CREATE VIEW sales_metastore.sales.sales_redacted AS
SELECT
  user_id,
  email,
  country,
  product,
  total
FROM sales_metastore.sales.sales_raw;
```

Python

```
spark.sql("CREATE VIEW sales_metastore.sales.sales_redacted AS "
  "SELECT "
  "  user_id, "
  "  email, "
  "  country, "
  "  product, "
  "  total "
  "FROM sales_metastore.sales.sales_raw")
```

R

```
library(SparkR)
```

```
sql(paste("CREATE VIEW sales_metastore.sales.sales_redacted AS ",
 "SELECT ",
 "  user_id, ",
 "  email, ",
 "  country, ",
 "  product, ",
 "  total ",
 "FROM sales_metastore.sales.sales_raw",
 sep = ""))
```

Scala

```
spark.sql("CREATE VIEW sales_metastore.sales.sales_redacted AS " +
 "SELECT " +
 "  user_id, " +
 "  email, " +
 "  country, " +
 "  product, " +
 "  total " +
 "FROM sales_metastore.sales.sales_raw")
```

Create a dynamic view

In Unity Catalog, you can use dynamic views to configure fine-grained access control, including:

- Security at the level of columns or rows.
- Data masking.

Note

Fine-grained access control using dynamic views is not available on clusters with **Single User access mode**.

Unity Catalog introduces the following functions, which allow you to dynamically limit which users can access a row, column, or record in a view:

- current_user(): Returns the current user's email address.

- is_account_group_member(): Returns TRUE if the current user is a member of a specific account-level group. Recommended for use in dynamic views against Unity Catalog data.
- is_member(): Returns TRUE if the current user is a member of a specific workspace-level group. This function is provided for compatibility with the existing Hive metastore. Avoid using it with views against Unity Catalog data, because it does not evaluate account-level group membership.

Azure Databricks recommends that you do not grant users the ability to read the tables and views referenced in the view.

The following examples illustrate how to create dynamic views in Unity Catalog.

Column-level permissions

With a dynamic view, you can limit the columns a specific user or group can access. In the following example, only members of the auditors group can access email addresses from the sales_raw table. During query analysis, Apache Spark replaces the CASE statement with either the literal string REDACTED or the actual contents of the email address column. Other columns are returned as normal. This strategy has no negative impact on the query performance.

SQL

```
-- Alias the field 'email' to itself (as 'email') to prevent the

-- permission logic from showing up directly in the column name results.

CREATE VIEW sales_redacted AS

SELECT

 user_id,

 CASE WHEN

  is_account_group_member('auditors') THEN email

  ELSE 'REDACTED'

 END AS email,

 country,

 product,

 total

FROM sales_raw
```

Python

```
# Alias the field 'email' to itself (as 'email') to prevent the

# permission logic from showing up directly in the column name results.

spark.sql("CREATE VIEW sales_redacted AS "
  "SELECT "
  "  user_id, "
  "  CASE WHEN "
  "    is_account_group_member('auditors') THEN email "
  "  ELSE 'REDACTED' "
  "  END AS email, "
  "  country, "
  "  product, "
  "  total "
  "FROM sales_raw")
```

R

```
library(SparkR)


# Alias the field 'email' to itself (as 'email') to prevent the

# permission logic from showing up directly in the column name results.

sql(paste("CREATE VIEW sales_redacted AS ",
  "SELECT ",
  "  user_id, ",
  "  CASE WHEN ",
  "    is_account_group_member('auditors') THEN email ",
  "  ELSE 'REDACTED' ",
  "  END AS email, ",
  "  country, ",
  "  product, ",
  "  total ",
  "FROM sales_raw",
  sep = ""))
```

Scala

```
// Alias the field 'email' to itself (as 'email') to prevent the
// permission logic from showing up directly in the column name results.
spark.sql("CREATE VIEW sales_redacted AS " +
  "SELECT " +
  "  user_id, " +
  "  CASE WHEN " +
  "    is_account_group_member('auditors') THEN email " +
  "  ELSE 'REDACTED' " +
  "  END AS email, " +
  "  country, " +
  "  product, " +
  "  total " +
  "FROM sales_raw")
```

Row-level permissions

With a dynamic view, you can specify permissions down to the row or field level. In the following example, only members of the managers group can view transaction amounts when they exceed $1,000,000. Matching results are filtered out for other users.

SQL

```
CREATE VIEW sales_redacted AS
 SELECT
   user_id,
   country,
   product,
   total
 FROM sales_raw
 WHERE
   CASE
     WHEN is_account_group_member('managers') THEN TRUE
     ELSE total <= 1000000
   END;
```

Python

```
spark.sql("CREATE VIEW sales_redacted AS "
  "SELECT "
  "  user_id, "
  "  country, "
  "  product, "
  "  total "
  "FROM sales_raw "
  "WHERE "
  "CASE "
  "  WHEN is_account_group_member('managers') THEN TRUE "
  "  ELSE total <= 1000000 "
  "END")
```

R

```
library(SparkR)

 sql(paste("CREATE VIEW sales_redacted AS ",
  "SELECT ",
  "  user_id, ",
  "  country, ",
  "  product, ",
  "  total ",
  "FROM sales_raw ",
  "WHERE ",
  "CASE ",
  "  WHEN is_account_group_member('managers') THEN TRUE ",
  "  ELSE total <= 1000000 ",
  "END",
  sep = ""))
```

Scala

```
spark.sql("CREATE VIEW sales_redacted AS " +
  "SELECT " +
  "  user_id, " +
  "  country, " +
  "  product, " +
  "  total " +
  "FROM sales_raw " +
  "WHERE " +
  "CASE " +
  "  WHEN is_account_group_member('managers') THEN TRUE " +
  "  ELSE total <= 1000000 " +
  "END")
```

Data masking

Because views in Unity Catalog use Spark SQL, you can implement advanced data masking by using more complex SQL expressions and regular expressions. In the following example, all users can analyze email domains, but only members of the auditors group can view a user's entire email address.

SQL

```
-- The regexp_extract function takes an email address such as
-- user.x.lastname@example.com and extracts 'example', allowing
-- analysts to query the domain name.
```

```sql
CREATE VIEW sales_redacted AS
SELECT
  user_id,
  region,
  CASE
    WHEN is_account_group_member('auditors') THEN email
    ELSE regexp_extract(email, '^.*@(.*)$', 1)
  END
  FROM sales_raw
```

Python

```python
# The regexp_extract function takes an email address such as
# user.x.lastname@example.com and extracts 'example', allowing
# analysts to query the domain name.

spark.sql("CREATE VIEW sales_redacted AS "
  "SELECT "
  "  user_id, "
  "  region, "
  "  CASE "
  "    WHEN is_account_group_member('auditors') THEN email "
  "    ELSE regexp_extract(email, '^.*@(.*)$', 1) "
  "  END "
  "  FROM sales_raw")
```

R

```r
library(SparkR)

# The regexp_extract function takes an email address such as
# user.x.lastname@example.com and extracts 'example', allowing
# analysts to query the domain name.


sql(paste("CREATE VIEW sales_redacted AS ",
 "SELECT ",
 "  user_id, ",
 "  region, ",
 "  CASE ",
 "    WHEN is_account_group_member('auditors') THEN email ",
 "    ELSE regexp_extract(email, '^.*@(.*)$', 1) ",
 "  END ",
 "  FROM sales_raw",
 sep = ""))
```

Scala

```
// The regexp_extract function takes an email address such as
// user.x.lastname@example.com and extracts 'example', allowing
// analysts to query the domain name.


spark.sql("CREATE VIEW sales_redacted AS " +
 "SELECT " +
 "  user_id, " +
 "  region, " +
 "  CASE " +
 "    WHEN is_account_group_member('auditors') THEN email " +
 "    ELSE regexp_extract(email, '^.*@(.*)$', 1) " +
 "  END " +
 "  FROM sales_raw")
```

Drop a view

You must be the view's owner to drop a view. To drop a view, run the following SQL command:

SQL

```sql
DROP VIEW IF EXISTS catalog_name.schema_name.view_name;
```

Apply tags

Tags are attributes containing keys and optional values that you can apply to different securable objects in Unity Catalog. Tagging is useful for organizing and categorizing different securable objects within a metastore. Using tags also simplifies search and discovery of your data assets.

Supported securable objects

Securable object tagging is currently supported on catalogs, schemas, tables, volumes, views, and registered models.

Requirements

To add tags to Unity Catalog securable objects, you must have the APPLY TAG privilege on the object, as well as the USE SCHEMA privilege on the object's parent schema and the USE CATALOG privilege on the object's parent catalog.

Constraints

The following is a list of tag constraints:

- You can assign a maximum of 20 tags to a single securable object.
- The maximum length of a tag is 255 characters.
- Special characters cannot be used in tag names.
- Searching Unity Catalog tables by tag name or value is supported with the exact match only.

Manage tags in Catalog Explorer

To manage securable object tags using the Catalog Explorer UI you must have at least the BROWSE privilege on the object.

1. Click **Catalog** in the sidebar.
2. Select a securable object to view the tag information.
3. Click **Add/Edit Tags** to manage tags for the current securable object. You can add and remove multiple tags simultaneously in the tag management modal.

Retrieve tag information with Information Schema tables

Each catalog created on Unity Catalog includes an `INFORMATION_SCHEMA`. This schema includes tables describing the objects known to the schema's catalog. You must have the appropriate privileges to view the schema information.

You can query the following to retrieve tag information:

- INFORMATION_SCHEMA.CATALOG_TAGS
- INFORMATION_SCHEMA.COLUMN_TAGS
- INFORMATION_SCHEMA.SCHEMA_TAGS
- INFORMATION_SCHEMA.TABLE_TAGS
- INFORMATION_SCHEMA.VOLUME_TAGS

Manage tags with SQL commands

Note

This feature is only available for Databricks Runtime versions of 13.3 and above.

Azure Databricks supports tagging SQL commands with catalogs, schemas, tables (views, materialized views, streaming tables), volumes, and table columns. For example, you can use the SET TAGS and UNSET TAGS clauses with the ALTER TABLE command to manage tags on a table.

Use tags to search and filter tables

To search and filter tables using tags:

1. Click the **Search** field in the top bar of the Azure Databricks workspace or use the keyboard shortcut Command-P.
2. Enter your search criteria. Search for tables in Unity Catalog by entering the assigned tag name or value.

You can also use the tag filter in the **Table** search tab to filter search results with existing tag names.

Capture and view data lineage using Unity Catalog

You can use Unity Catalog to capture runtime data lineage across queries run on Azure Databricks. Lineage is supported for all languages and is captured down to the column level. Lineage data includes notebooks, workflows, and dashboards related to the query. Lineage can be visualized in Catalog Explorer in near real-time and retrieved with the Databricks REST API.

Note

You can also view and query lineage data using the lineage system tables (Public Preview).

Lineage is aggregated across all workspaces attached to a Unity Catalog metastore. This means that lineage captured in one workspace is visible in any other workspace sharing that metastore. Users must have the correct permissions to view the lineage data. Lineage data is retained for 1 year.

Requirements

The following are required to capture data lineage using Unity Catalog:

- The workspace must have Unity Catalog enabled.
- Tables must be registered in a Unity Catalog metastore.
- Queries must use the Spark DataFrame (for example, Spark SQL functions that return a DataFrame) or Databricks SQL interfaces. For examples of Databricks SQL and PySpark queries, see: .lineage-example.dbc (github.com)
- To view the lineage of a table or view, users must have at least the BROWSE privilege on the table's or view's parent catalog.
- To view lineage information for notebooks, workflows, or dashboards, users must have permissions on these objects as defined by the access control settings in the workspace.
- To view lineage for a Unity Catalog-enabled pipeline, you must have CAN_VIEW permissions on the pipeline.
- You might need to update your outbound firewall rules to allow for connectivity to the Event Hub endpoint in the Azure Databricks control plane. Typically this applies if your Azure Databricks workspace is deployed in your own VNet (also known as VNet injection).

Limitations

- Streaming between Delta tables is supported only in Databricks Runtime 11.3 LTS or above.

- Because lineage is computed on a 1-year rolling window, lineage collected more than 1 year ago is not displayed. For example, if a job or query reads data from table A and writes to table B, the link between table A and table B is displayed for only 1 year.

You can filter lineage data by time frame. When "All lineage" is selected, lineage data collected starting in June 2023 is displayed.

- Workflows that use the Jobs API runs submit request are unavailable when viewing lineage. Table and column level lineage is still captured when using the runs submit request, but the link to the run is not captured.
- Unity Catalog captures lineage to the column level as much as possible. However, there are some cases where column-level lineage cannot be captured.
- Column lineage is only supported when both the source and target are referenced by table name (Example: select * from <catalog>.<schema>.<table>). Column lineage cannot be captured if the source or the target are addressed by path (Example: select * from delta."s3://<bucket>/<path>").
- Data lineage doesn't capture reads on tables with column masks.
- If a table is renamed, lineage is not captured for the renamed table.
- If you use Spark SQL dataset checkpointing, lineage is not captured.
- Unity Catalog captures lineage from Delta Live Tables pipelines in most cases. However, in some instances, complete lineage coverage cannot be guaranteed, such as when pipelines use the APPLY CHANGES API or TEMPORARY tables.

Examples

Note

- The following examples use the catalog name lineage_data and the schema name lineagedemo. To use a different catalog and schema, change the names used in the examples.
- To complete this example, you must have CREATE and USE SCHEMA privileges on a schema. A metastore admin, catalog owner, or schema owner can grant these privileges. For example, to give all users in the group 'data_engineers' permission to create tables in the lineagedemo schema in the lineage_data catalog, a user with one of the above privileges or roles can run the following queries:

SQL

```
CREATE SCHEMA lineage_data.lineagedemo;

GRANT USE SCHEMA, CREATE on SCHEMA lineage_data.lineagedemo to
`data_engineers`;
```

Capture and explore lineage

To capture lineage data, use the following steps:

1. Go to your Azure Databricks landing page, click **New** in the sidebar, and select **Notebook** from the menu.
2. Enter a name for the notebook and select **SQL** in **Default Language**.
3. In **Cluster**, select a cluster with access to Unity Catalog.
4. Click **Create**.
5. In the first notebook cell, enter the following queries:

SQL

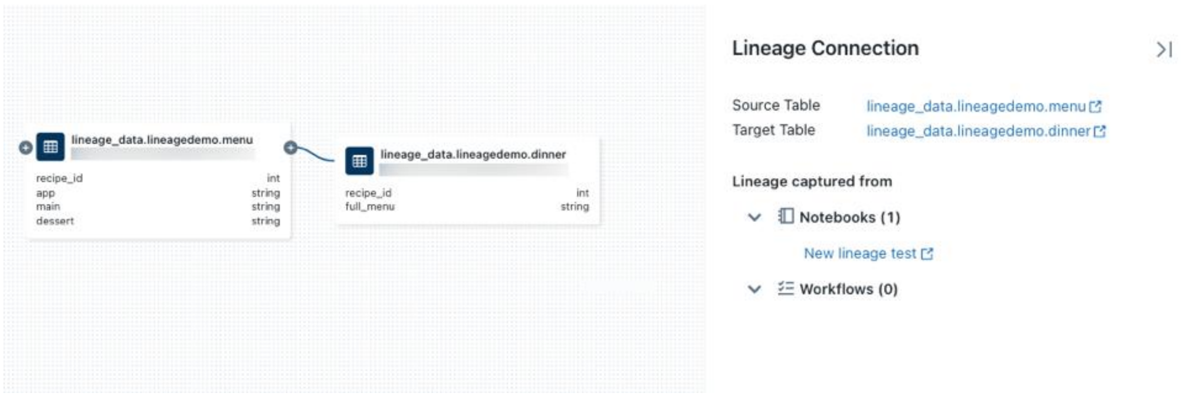
```
CREATE TABLE IF NOT EXISTS
 lineage_data.lineagedemo.menu (
  recipe_id INT,
  app string,
  main string,
  dessert string
 );


INSERT INTO lineage_data.lineagedemo.menu
  (recipe_id, app, main, dessert)
VALUES
  (1,"Ceviche", "Tacos", "Flan"),
  (2,"Tomato Soup", "Souffle", "Creme Brulee"),
  (3,"Chips","Grilled Cheese","Cheesecake");


CREATE TABLE
```

```
 lineage_data.lineagedemo.dinner
AS SELECT
 recipe_id, concat(app," + ", main," + ",dessert)
AS
 full_menu
FROM
 lineage_data.lineagedemo.menu
```

6. To run the queries, click in the cell and press **shift+enter** or click ▶▾ and select **Run Cell**.

To use Catalog Explorer to view the lineage generated by these queries, use the following steps:

1. In the **Search** box in the top bar of the Azure Databricks workspace, enter lineage_data.lineagedemo.dinner and click **Search lineage_data.lineagedemo.dinner in Databricks**.
2. Under **Tables**, click the dinner table.
3. Select the **Lineage** tab. The lineage panel appears and displays related tables (for this example it's the menu table).
4. To view an interactive graph of the data lineage, click **See Lineage Graph**. By default, one level is displayed in the graph. You can click on the ⊕ icon on a node to reveal more connections if they are available.
5. Click on an arrow connecting nodes in the lineage graph to open the **Lineage connection** panel. The **Lineage connection** panel shows details about the connection, including source and target tables, notebooks, and workflows.



lineage_data.lineagedemo.menu

```
recipe_id     int
app           string
main          string
dessert       string
```

lineage_data.lineagedemo.dinner

```
recipe_id     int
full_menu     string
```

Lineage Connection

Source Table lineage_data.lineagedemo.menu

Target Table lineage_data.lineagedemo.dinner

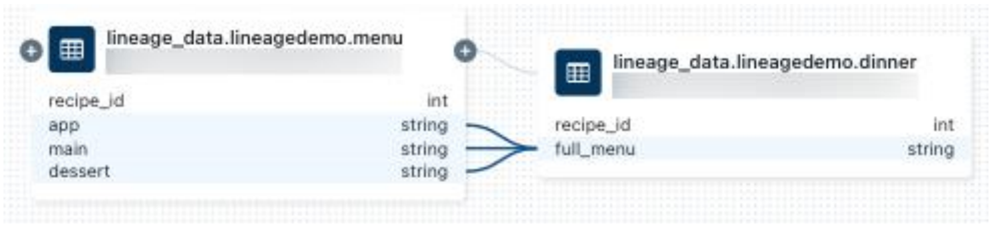
Lineage captured from

Notebooks (1)

New lineage test

Workflows (0)

6. To show the notebook associated with the dinner table, select the notebook in the **Lineage connection** panel or close the lineage graph and click **Notebooks**. To open the notebook in a new tab, click on the notebook name.
7. To view the column-level lineage, click on a column in the graph to show links to related columns. For example, clicking on the 'full_menu' column shows the upstream columns the column was derived from:



To demonstrate creating and viewing lineage with a different language, for example, Python, use the following steps:

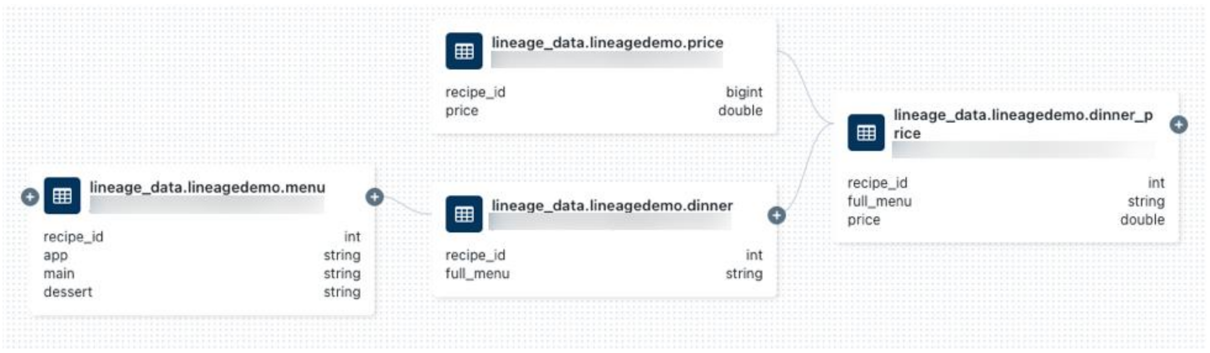
1. Open the notebook you created previously, create a new cell, and enter the following Python code:

Python

```
%python

from pyspark.sql.functions import rand, round

df = spark.range(3).withColumn("price",
round(10*rand(seed=42),2)).withColumnRenamed("id","recipe_id")


df.write.mode("overwrite").saveAsTable("lineage_data.lineagedemo.price")


dinner = spark.read.table("lineage_data.lineagedemo.dinner")

price = spark.read.table("lineage_data.lineagedemo.price")


dinner_price = dinner.join(price, on="recipe_id")

dinner_price.write.mode("overwrite").saveAsTable("lineage_data.lineagedemo.dinner_p
rice")
```

2. Run the cell by clicking in the cell and pressing **shift+enter** or clicking ▶ and selecting **Run Cell**.

3. In the **Search** box in the top bar of the Azure Databricks workspace, enter lineage_data.lineagedemo.price and click **Search lineage_data.lineagedemo.price in Databricks**.
4. Under **Tables**, click the price table.
5. Select the **Lineage** tab and click **See Lineage Graph**. Click on the ⊕ icons to explore the data lineage generated by the SQL and Python queries.



6. Click on an arrow connecting nodes in the lineage graph to open the **Lineage connection** panel. The **Lineage connection** panel shows details about the connection, including source and target tables, notebooks, and workflows.

Capture and view workflow lineage

Lineage is also captured for any workflow that reads or writes to Unity Catalog. To demonstrate viewing lineage for an Azure Databricks workflow, use the following steps:

1. Click ⊕ **New** in the sidebar and select **Notebook** from the menu.
2. Enter a name for the notebook and select **SQL** in **Default Language**.
3. Click **Create**.
4. In the first notebook cell, enter the following query:

SQL

```
SELECT * FROM lineage_data.lineagedemo.menu
```

5. Click **Schedule** in the top bar. In the schedule dialog, select **Manual**, select a cluster with access to Unity Catalog, and click **Create**.
6. Click **Run now**.
7. In the **Search** box in the top bar of the Azure Databricks workspace, enter lineage_data.lineagedemo.menu and click **Search lineage_data.lineagedemo.menu in Databricks**.
8. Under **Tables View all tables**, click the menu table.

9. Select the **Lineage** tab, click **Workflows**, and select the **Downstream** tab. The job name appears under **Job Name** as a consumer of the menu table.

Capture and view dashboard lineage

To demonstrate viewing lineage for a SQL dashboard, use the following steps:

1. Go to your Azure Databricks landing page and open Catalog Explorer by clicking **Catalog** in the sidebar.
2. Click on the catalog name, click **lineagedemo**, and select the menu table. You can also use the **Search tables** text box in the top bar to search for the menu table.
3. Click **Actions > Create a quick dashboard**.
4. Select columns to add to the dashboard and click **Create**.
5. In the **Search** box in the top bar of the Azure Databricks workspace, enter lineage_data.lineagedemo.menu and click **Search lineage_data.lineagedemo.menu in Databricks**.
6. Under **Tables View all tables**, click the menu table.
7. Select the **Lineage** tab and click **Dashboards**. The dashboard name appears under **Dashboard Name** as a consumer of the menu table.

Lineage permissions

Lineage graphs share the same permission model as Unity Catalog. If a user does not have the BROWSE or SELECT privilege on a table, they cannot explore the lineage. Additionally, users can only see notebooks, workflows, and dashboards that they have permission to view. For example, if you run the following commands for a non-admin user userA:

SQL

```
GRANT USE SCHEMA on lineage_data.lineagedemo to `userA@company.com`;

GRANT SELECT on lineage_data.lineagedemo.menu to `userA@company.com`;
```

When userA views the lineage graph for the lineage_data.lineagedemo.menu table, they will see the menu table. They will not be able to see information about associated tables, such as the downstream lineage_data.lineagedemo.dinner table.
The dinner table is displayed as a masked node in the display to userA,
and userA cannot expand the graph to reveal downstream tables from tables they do not have permission to access.

If you run the following command to grant the BROWSE permission to a non-admin user userB:

SQL

```
GRANT BROWSE on lineage_data to `userA@company.com`;
```

userB can now view the lineage graph for any table in the lineage_data schema.

Delete lineage data

Warning

The following instructions delete all objects stored in Unity Catalog. Use these instructions only if necessary. For example, to meet compliance requirements.

To delete lineage data, you must delete the metastore managing the Unity Catalog objects.

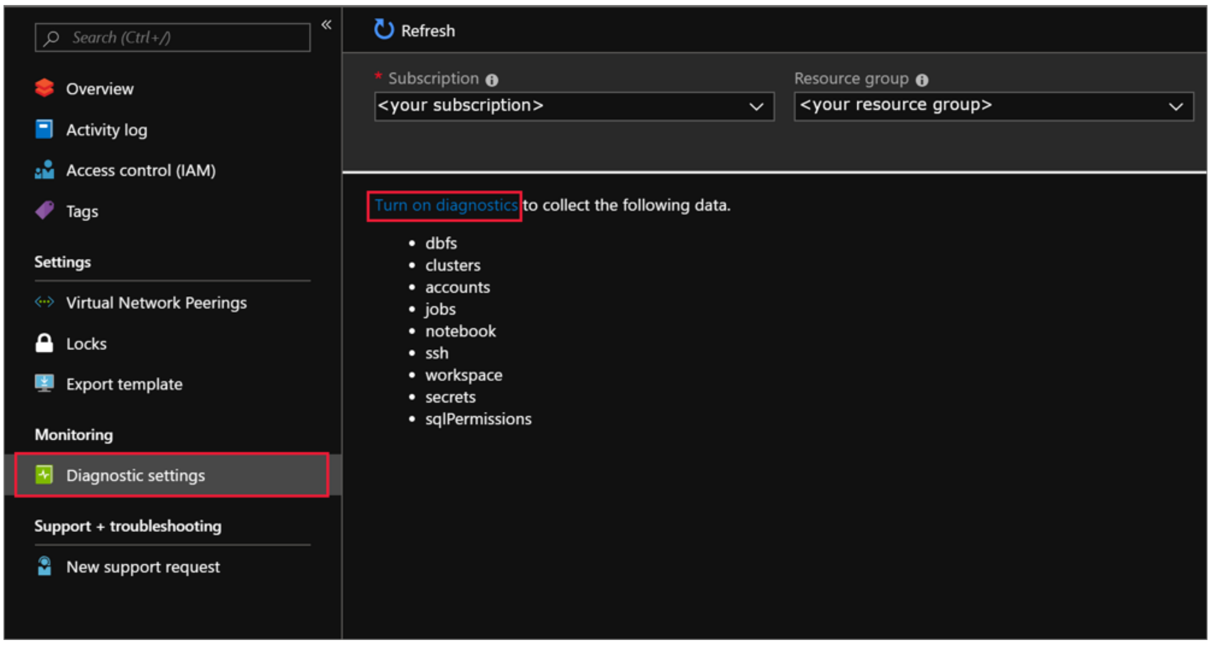
Audit Unity Catalog events

Configure diagnostic logs

To access diagnostic logs for Unity Catalog events, you must enable and configure diagnostic logs for each workspace in your account.

Configure diagnostic log delivery

1. Log in to the Azure portal as an Owner or Contributor for the Azure Databricks workspace and click your Azure Databricks Service resource.
2. In the Monitoring section of the sidebar, click the **Diagnostic settings** tab.
3. Click **Turn on diagnostics**.



Turn on diagnostics to collect the following data.

- dbfs
- clusters
- accounts
- jobs
- notebook
- ssh
- workspace
- secrets
- sqlPermissions

4. On the Diagnostic settings page, provide the following configuration:

Name

Enter a name for the logs to create.

Archive to a storage account

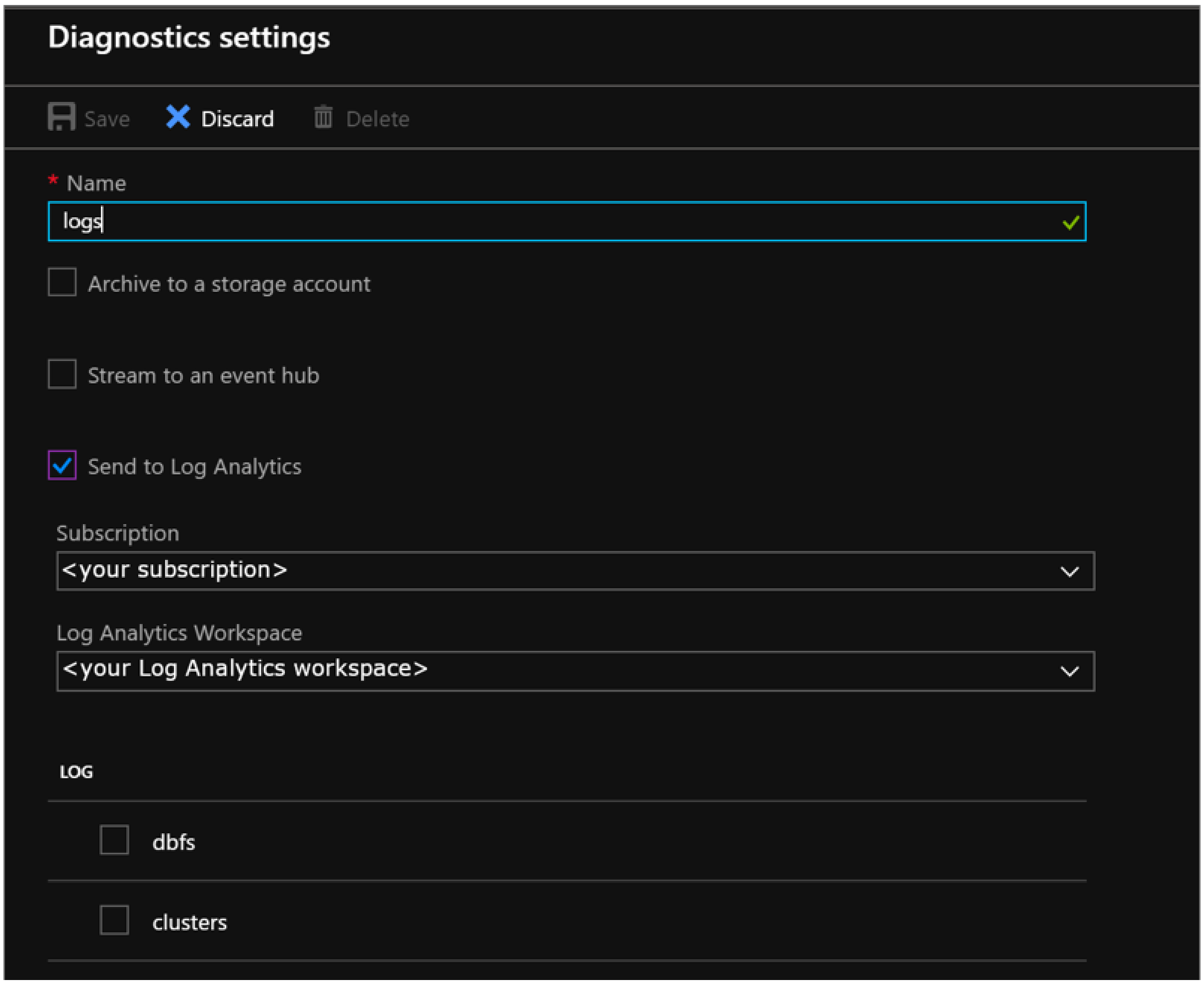
To use this option, you need an existing storage account to connect to. To create a new storage account in the portal, and follow the instructions to create an Azure Resource Manager, general-purpose account. Then return to this page in the portal to select your storage account. It might take a few minutes for newly created storage accounts to appear in the drop-down menu.

Stream to an event hub

To use this option, you need an existing Azure Event Hubs namespace and event hub to connect to. To create an Event Hubs namespace. Then return to this page in the portal to select the Event Hubs namespace and policy name.

Send to Log Analytics

To use this option, either use an existing Log Analytics workspace or create a new one by following the steps to Create a new workspace in the portal.



Name: logs

☐ Archive to a storage account

☐ Stream to an event hub

☑ Send to Log Analytics

Subscription: <your subscription>

Log Analytics Workspace: <your Log Analytics workspace>

LOG

☐ dbfs

☐ clusters

5. Choose the services you want diagnostic logs for.
6. Select **Save**.
7. If you receive an error that says “Failed to update diagnostics for <workspace name>. The subscription <subscription id> is not registered to use microsoft.insights,” follow the Troubleshoot Azure Diagnostics instructions to register the account and then retry this procedure.

8. If you want to change how your diagnostic logs are saved at any point in the future, return to this page to modify the diagnostic log settings for your account.

Note

Enable logging using PowerShell

1. Start an Azure PowerShell session and sign in to your Azure account with the following command:

PowerShell

```
Connect-AzAccount
```

To sign in to your Azure account as a user, To sign in to your Azure account as a service principal.

If you do not have Azure Powershell installed already, use the following commands to install Azure PowerShell.

PowerShell

```
Install-Module -Name Az -AllowClobber
```

2. In the pop-up browser window, enter your Azure account user name and password. Azure PowerShell gets all of the subscriptions that are associated with this account, and by default, uses the first one.

If you have more than one subscription, you might have to specify the specific subscription that was used to create your Azure Key Vault. To see the subscriptions for your account, type the following command:

PowerShell

```
Get-AzSubscription
```

To specify the subscription that's associated with the Azure Databricks account that you're logging, type the following command:

PowerShell

```
Set-AzContext -SubscriptionId <subscription ID>
```

3. Set your Log Analytics resource name to a variable named logAnalytics, where ResourceName is the name of the Log Analytics workspace.

PowerShell

```
$logAnalytics = Get-AzResource -ResourceGroupName <resource group name> -
ResourceName <resource name> -ResourceType
"Microsoft.OperationalInsights/workspaces"
```

4. Set the Azure Databricks service resource name to a variable named databricks, where ResourceName is the name of the Azure Databricks service.

PowerShell

```
$databricks = Get-AzResource -ResourceGroupName <your resource group name> -ResourceName <your Azure Databricks service name> -ResourceType "Microsoft.Databricks/workspaces"
```

5. To enable logging for Azure Databricks, use the **New-AzDiagnosticSetting** cmdlet with variables for the new storage account, Azure Databricks service, and the category to enable for logging. Run the following command and set the -Enabled flag to $true:

PowerShell

```
New-AzDiagnosticSetting -ResourceId $databricks.ResourceId -WorkspaceId $logAnalytics.ResourceId -Enabled $true -name "<diagnostic setting name>" -Category <comma separated list>
```

Enable logging by using Azure CLI

1. Open PowerShell.
2. Use the following command to connect to your Azure account:

PowerShell

```
az login
```

3. Run the following diagnostic setting command:

PowerShell

```
az monitor diagnostic-settings create --name <diagnostic name>

--resource-group <log analytics workspace resource group>

--workspace <log analytics name or object ID>

--resource <target resource object ID>

--logs '[

{

 \"category\": <category name>,

  \"enabled\": true

}
```

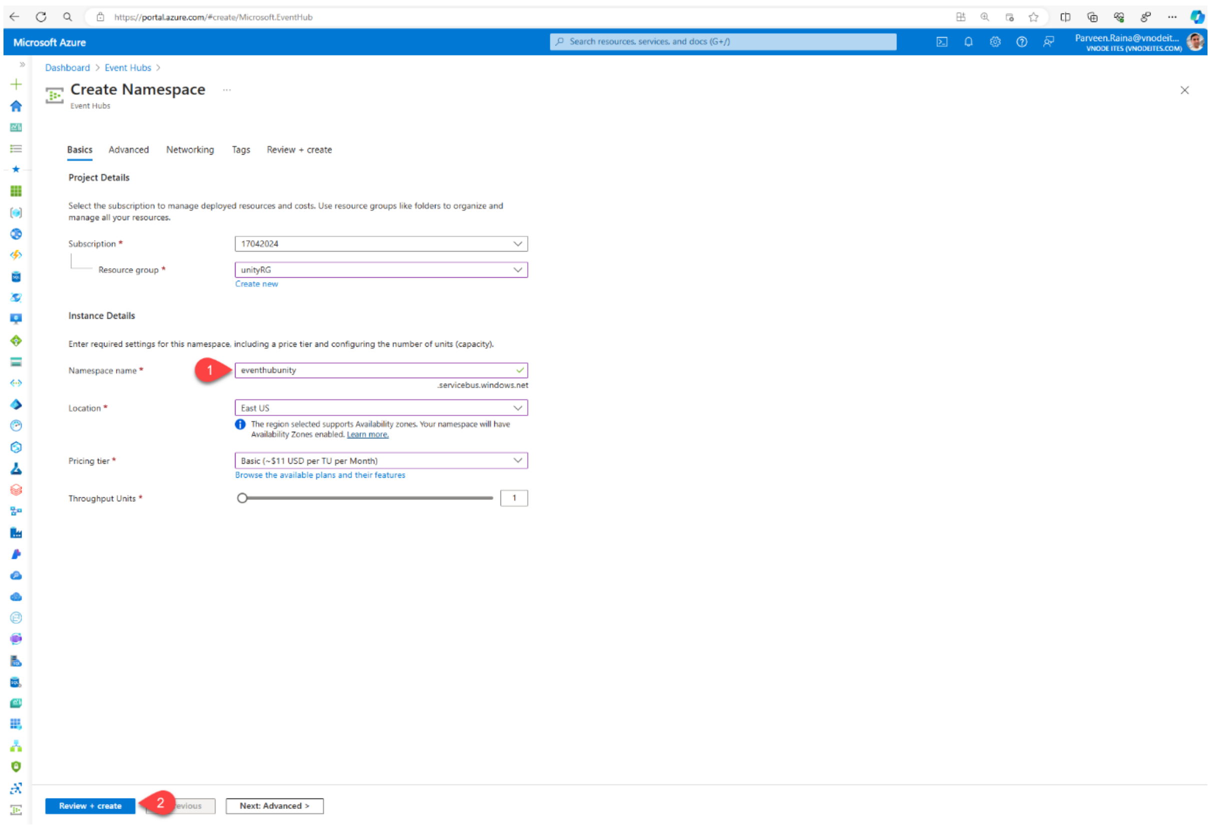
]

Important

Azure Databricks does not log Unity Catalog events that take place only at the account level. Only events that are associated with a workspace are logged.

To create tables of logged Unity Catalog events that you can query:

1. Create or reuse an Event Hubs namespace.



Project Details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * 17042024

Resource group * unityRG

Create new

Instance Details

Enter required settings for this namespace, including a price tier and configuring the number of units (capacity).

Namespace name * eventhubunity .servicebus.windows.net

Location * East US

The region selected supports Availability zones. Your namespace will have Availability Zones enabled. Learn more.

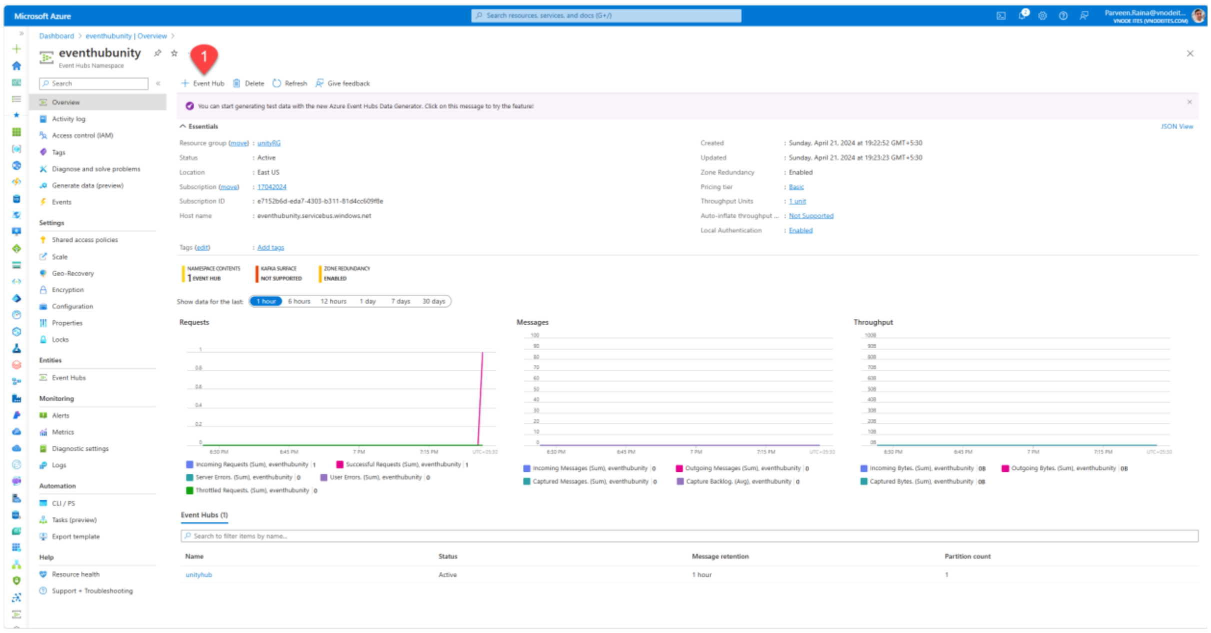
Pricing tier * Basic (~$11 USD per TU per Month)

Browse the available plans and their features

Throughput Units * 1

This namespace must be in the same region as the Azure Databricks workspace.

2. Create an event hub inside the namespace.



eventhubunity
Event Hubs Namespace

You can start generating test data with the new Azure Event Hubs Data Generator. Click on this message to try the feature!

Essentials

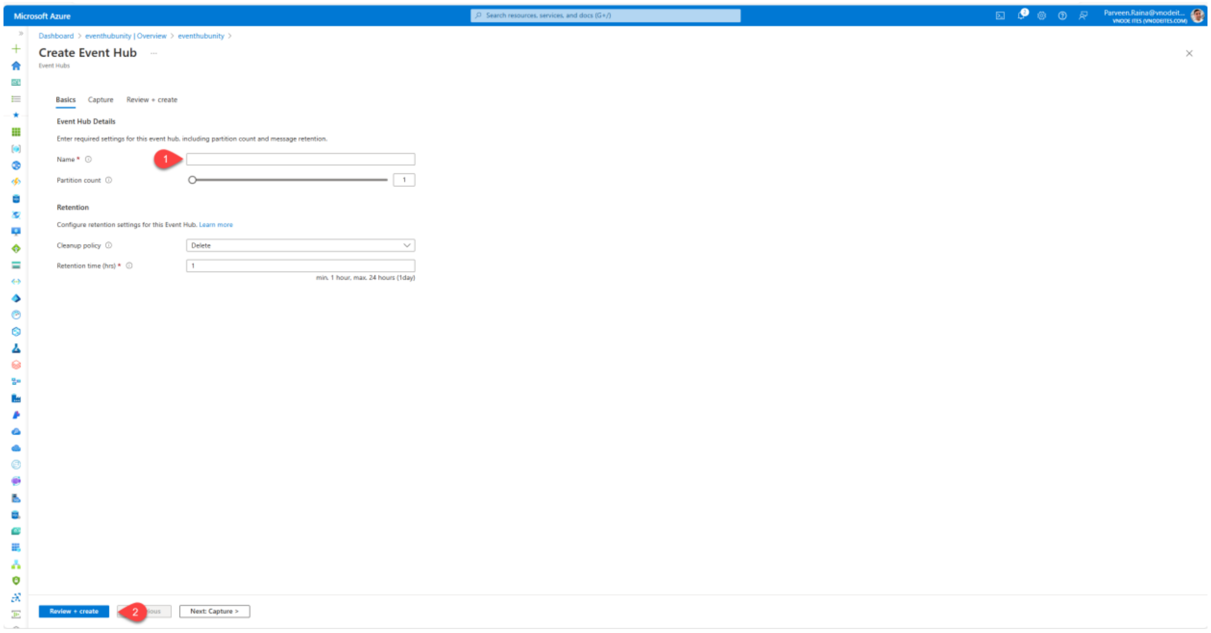
Resource group (move)	: unityRG
Status	: Active
Location	: East US
Subscription (move)	: 17042024
Subscription ID	: e7152b6d-eda7-4303-b311-81d4cc609f8e
Host name	: eventhubunity.servicebus.windows.net
Tags (edit)	: Add tags
Created	: Sunday, April 21, 2024 at 19:22:52 GMT+5:30
Updated	: Sunday, April 21, 2024 at 19:23:23 GMT+5:30
Zone Redundancy	: Enabled
Pricing tier	: Basic
Throughput Units	: 1 unit
Auto-inflate throughput ...	: Not Supported
Local Authentication	: Enabled

NAMESPACE CONTENTS 1 EVENT HUB
KAFKA SURFACE NOT SUPPORTED
ZONE REDUNDANCY ENABLED

Event Hubs (1)

Search to filter items by name...

Name	Status	Message retention	Partition count
unityhub	Active	1 hour	1


Microsoft Azure
Search resources, services, and docs (G+/)
Parveen.Raina@vnodeit...
VNODE ITES (VNODEITES.COM)
Dashboard > eventhubunity | Overview > eventhubunity >
Basics
Capture
Review + create
1
2
Review + create
Next: Capture >

Create Event Hub
Event Hubs

Event Hub Details

Enter required settings for this event hub, including partition count and message retention.

Name *	
Partition count	1

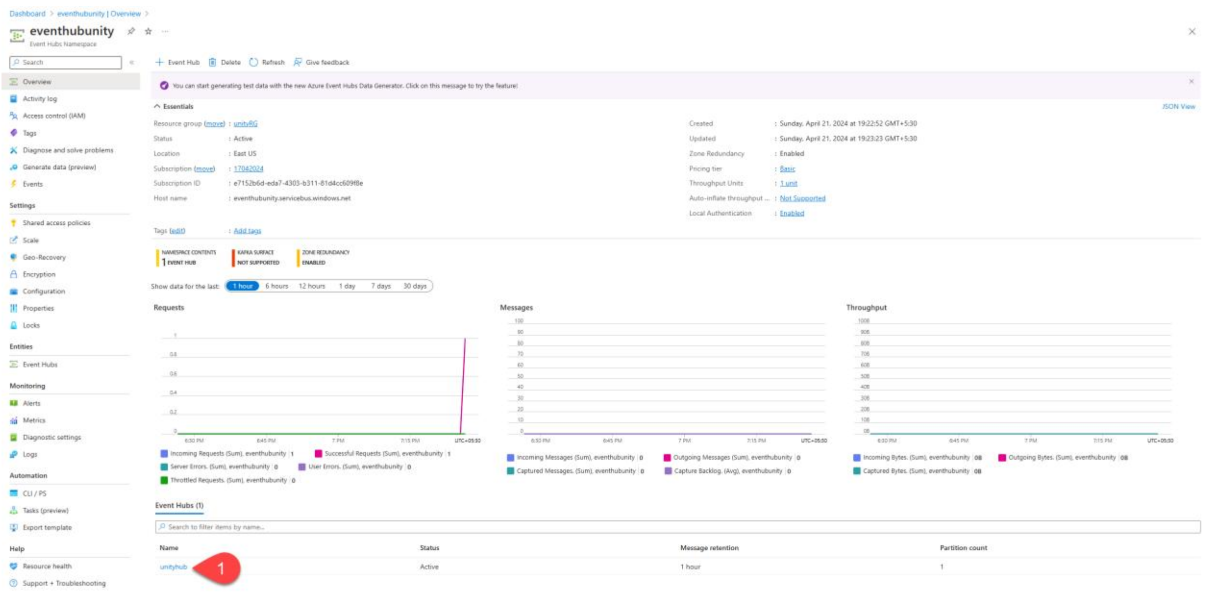
Retention

Configure retention settings for this Event Hub. Learn more

Cleanup policy	Delete
Retention time (hrs) *	1

min. 1 hour, max. 24 hours (1day)

3. Copy the connection string for the Event Hub.



You can start generating test data with the new Azure Event Hubs Data Generator. Click on this message to try the feature!

Essentials

Resource group (move)	: unityRG
Status	: Active
Location	: East US
Subscription (move)	: 17042024
Subscription ID	: e7152b6d-eda7-4303-b311-81d4cc609f8e
Host name	: eventhubunity.servicebus.windows.net
Tags (edit)	: Add tags
Created	: Sunday, April 21, 2024 at 19:22:52 GMT+5:30
Updated	: Sunday, April 21, 2024 at 19:23:23 GMT+5:30
Zone Redundancy	: Enabled
Pricing tier	: Basic
Throughput Units	: 1 unit
Auto-inflate throughput ...	: Not Supported
Local Authentication	: Enabled

NAMESPACE CONTENTS 1 EVENT HUB
KAFKA SURFACE NOT SUPPORTED
ZONE REDUNDANCY ENABLED

Incoming Requests (Sum), eventhubunity 1
Successful Requests (Sum), eventhubunity 1
Server Errors. (Sum), eventhubunity 0
User Errors. (Sum), eventhubunity 0
Throttled Requests. (Sum), eventhubunity 0

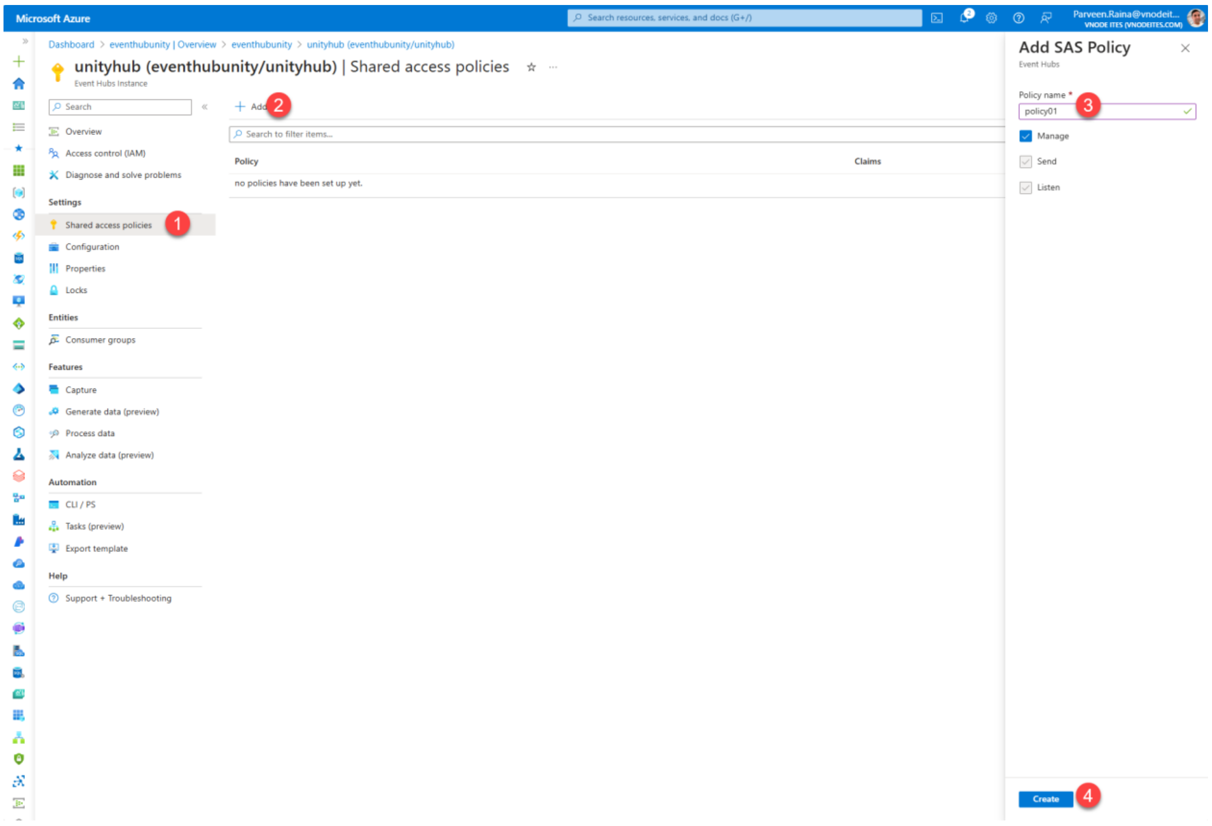
Incoming Messages (Sum), eventhubunity 0
Outgoing Messages (Sum), eventhubunity 0
Captured Messages. (Sum), eventhubunity 0
Capture Backlog. (Avg), eventhubunity 0

Incoming Bytes. (Sum), eventhubunity 0B
Outgoing Bytes. (Sum), eventhubunity 0B
Captured Bytes. (Sum), eventhubunity 0B

Event Hubs (1)

Search to filter items by name...

Name	Status	Message retention	Partition count
unityhub	Active	1 hour	1



Policy	Claims
no policies have been set up yet.	

Add SAS Policy

Event Hubs

Policy name * policy01

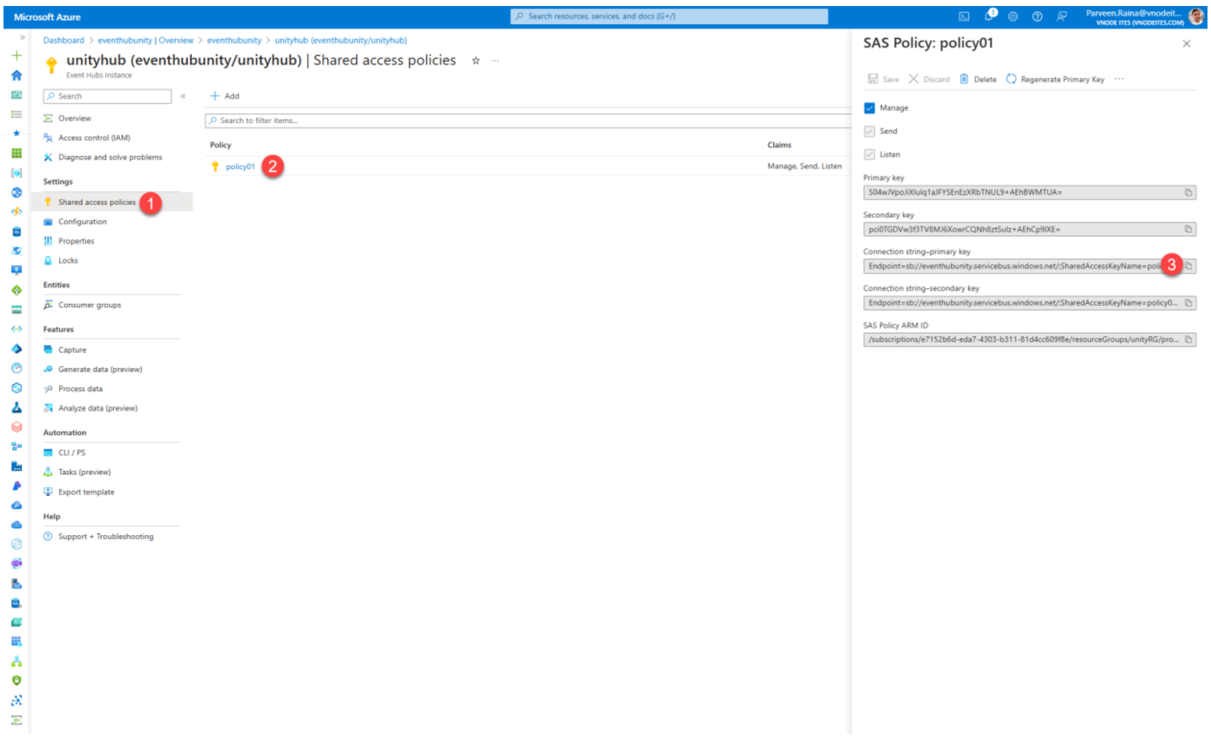
[✓] Manage

[✓] Send

[✓] Listen

Create

Parveen KR MC ID: 7124506



Policy	Claims
policy01	Manage, Send, Listen

[✓] Manage
[✓] Send
[✓] Listen

Primary key: S04wJVpoJiXIulq1aJFYSEnEzXRbTNUL9+AEhBWMTUA=

Secondary key: pci0TGDVw3f3TV8MJ6XowrCQNh8ztSulz+AEhCp9IXE=

Connection string–primary key: Endpoint=sb://eventhubunity.servicebus.windows.net/;SharedAccessKeyName=poli

Connection string–secondary key: Endpoint=sb://eventhubunity.servicebus.windows.net/;SharedAccessKeyName=policy0...

SAS Policy ARM ID: /subscriptions/e7152b6d-eda7-4303-b311-81d4cc609f8e/resourceGroups/unityRG/pro...

The policy needs only the Listen permission. The connection string should begin with Endpoint=sb://.

4. Store the connection string as a secret in the Azure Databricks workspace

5. Enable diagnostic logs for the workspace in which you will process the diagnostic logs.

Select the following options:

- Destination: **Stream to an event hub**
- The Event Hubs namespace and event hub that you created in steps 1 and 2.
- Log category: **unityCatalog**

6. Create a cluster that uses the **Single User** access mode.

7. Import the following example notebook into your workspace and attach it to the cluster you just created.

Audit log report for Unity Catalog events

azure-diagnostic-logs-etl-unity-catalog.ipynb (github.com)

8. Fill in the fields in the second notebook cell:

- <catalog>: catalog where you want to store the audit tables (catalog must already exist). Make sure that you have USE
  CATALOG and CREATE privileges on it.
- <database>: database (schema) where you want to store the audit tables (will be created if doesn't already exist). If it does already exist, make sure that you have USE SCHEMA and CREATE privileges on it.
- <eh-ns-name>: name of the Event Hubs namespace that contains the event hub
- <eh-topic-name>: name of the event hub (topic)
- <secret-scope-name>: name of the secret scope for the secret that contains the Event Hubs connection string
- <secret-name>: name of the secret that contains the Event Hubs connection string
- <sink-path>: the DBFS path to Spark checkpoints; for example /tmp/unity-audit-logs.

9. Run the notebook to create the audit logging tables.