Unity Catalog (UC) Upgrade Guide

Introduction

Unity Catalog brings a new level of Data Governance to Databricks. Utilizing Unity Catalog for existing workspaces requires a number of upgrade tasks to be completed. This guidance is meant for a Databricks account administrator to perform.

The upgrade tasks include (but are not limited to)

1. **Activating Unity Catalog**
    a. Setting up your **users** with Unity Catalog
        i. Set up SCIM
        ii. Elevate groups at the account level
    b. Setting up your **workspace** with Unity Catalog
        i. Set up cloud permissions
        ii. Set up a metastore
        iii. Update your workspace assets
2. **Upgrading to Use Unity Catalog**
    a. Design your **catalog structure**
        i. Designing information layout
        ii. Multi-region design
    b. Upgrading your **tables** for Unity Catalog
        i. Upgrade (in place) external tables.
        ii. Upgrade managed tables
        iii. Upgrading hive metastores
    c. Setting up your **data access control** with Unity Catalog
        i. Upgrading data access control
        ii. Setting up access control
    d. Setting up your **clusters**, **workloads,** and **downstream tools**
        i. Upgrading clusters
        ii. Migrating jobs and notebooks
        iii. Upgrading your existing downstream tools for Unity Catalog
            1. Autoloader/Streaming
            2. ML workflows
            3. Databricks SQL (DBSQL)
            4. Fivetran
            5. DBT
    e. Setting up **Delta Sharing** (optional)
        i. Set up Delta Sharing within Unity Catalog
    f. Extra resources

i. Setting up new workspaces with UC using Terraform
    1. AWS

Pre-req

- You must be a Databricks account admin.
- In AWS, you must be on the Databricks E2 account.
    - All new Databricks accounts and most existing accounts are now E2. If you are unsure which account type you have, contact your Databricks representative.
- Your Databricks account must be on the Premium plan or above.
- In AWS, you must have the ability to create S3 buckets, IAM roles, IAM policies, and cross-account trust relationships. In Azure, you have the ability to create ADLS Gen 2 storage accounts, IAM roles, IAM policies, and service principals.
- You must have at least one workspace that you want to use with Unity Catalog. See Create and manage workspaces using the account console.
- Take the UC Databricks Academy course - Data Access Control in Unity Catalog - which you can get access through your Databricks representative OR watch this Data + AI Summit talk on Unity Catalog for background on the product.
- Your Databricks representative has done a qualification session with you to make sure this is the right upgrade for you.

Overview of Unity Catalog

Unity Catalog (UC) represents improved governance capabilities within Databricks by
- moving data access control to the metastore level rather than the cluster/warehouse level
- allowing you to manage users/groups/service principals across workspaces
- adding lineage capabilities
- perform data search and discovery
- UC will unlock even more capabilities down the line such as ABAC
- UC also unlocks Delta Sharing which allows you to do secure in-place data sharing for data in Delta Lake to any tool that supports Delta Lake.

Below are some resources to help give a better understanding of this Databricks product.

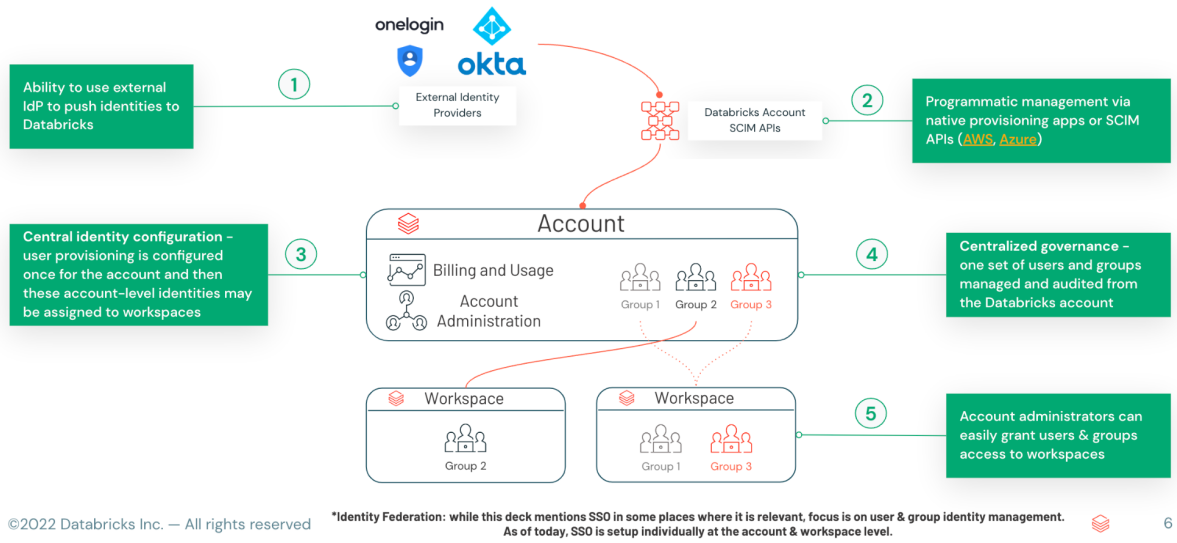
- AWS
- Azure

For more information, check out this talk on UC and this talk on Delta Sharing from the Data + AI Summit.

Step 1 - Setting up Groups with Unity Catalog

Set up SCIM

With Unity Catalog the workspace will use Identity Federation. See the picture below for Account Identity Information overview.

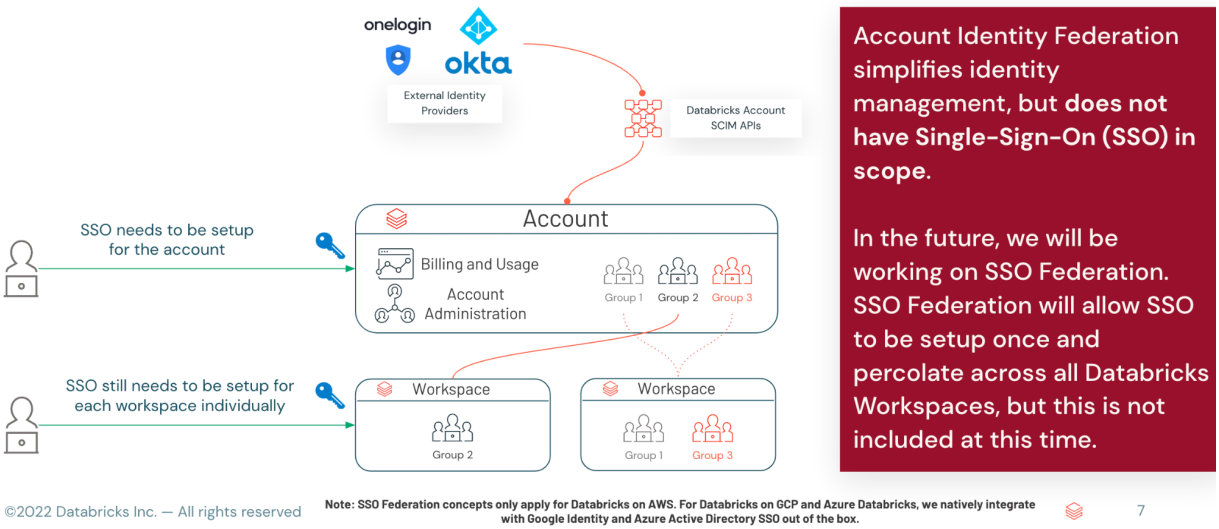


Workspace custom groups will not be supported in the future. SCIM providers have to be set at the account level.

Best Practice

- Use SCIM at the Account Console to provide all Principals to Unity Catalog and Databricks Workspaces
- Groups should be defined via SCIM and be consistent with your organizational group definitions.
- Do not use SCIM at the Workspace level at all.
    - If SCIM connectors are currently configured, once Identity Federation is enabled for your Databricks Account you should disable them.
    - Identity Federation will be by default turned on for all workspaces with metastores assigned.

Note: Identity Federation is not SSO Federation. See the picture below of where SSO needs to be set up separately from the identity federation steps here.



Setting up Groups at the account level

When workspaces are upgraded users are upgraded with the workspaces, however, groups are not. Therefore, follow these instructions to add users & groups. Note that the link is for Azure but AWS instructions are similar. Below are some definitions and links to help you in setting up/designing your groups and principal services.

Users

- User identities recognized by Databricks and represented by email addresses
- Can either be "local" to Databricks only (for AWS non-SSO) or can come from a remote identity provider via SAML, AAD, OIDC

Service Principals

- Identities for use with jobs, automated tools, and systems such as scripts, apps, and CI/CD platforms, represented by a uuid
- On Azure they come from AAD and on AWS they're Databricks-managed.

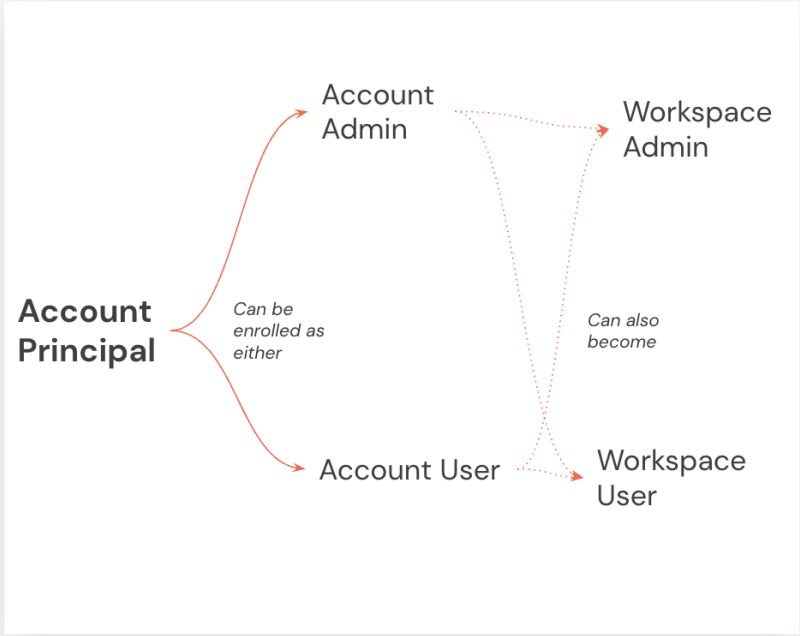
Groups

- A collection of identities used by admins to manage group access to workspaces, data, and other securable objects. All Databricks identities can be assigned as members of groups
- Can be "local" Databricks managed or synced in via SCIM
- Special groups:
    - Account-level: "Account users"
    - Workspace-level: "admins" and "users"

Account Level Identity Roles

When a new **principal** is added to the Account Console, this can be enrolled as an **Account Admin** (ability to manage workspaces, identities, etc.), or an **Account User** (only able to see other users, and workspaces they're assigned to).

Both these roles can be assigned to **Workspaces** - either as a **Workspace Admin** (ability to manage an individual workspace), or as a **Workspace User** (has the ability to use the workspace, and any permissions they're granted within the workspace itself).



Best Practices

- Avoid manual changes, i.e. adding users manually or manual modifications to group memberships.
- Use Groups for assigning access to grants for users, avoid assigning access to users directly.
- Use Service Principals for running jobs
- Use Groups for owning data and securable objects
- Think twice before providing any users with direct MODIFY access to production tables, this should be reserved for service principals.

Step 2 - Setting up your Workspace with Unity Catalog

Setting up your cloud permissions

Choose the below links with which cloud you are setting up UC:
AWS - Set up your IAM roles and S3 buckets for Unity Catalog.
Azure - Configure & grant access to Azure storage and your metastore

Metastores

Once Unity Catalog is set up the next step is creating the Metastores. Metastores are the physical store for Metadata and Governance in UC. The Unity Catalog Metastore is where Catalogs, Databases, Tables, Credentials and External locations are defined.

All workspaces assigned to a given Metastore will be able to access the same data catalogs once permissions are granted for the catalogs, databases, and tables within that metatore that a metastore admin wishes to grant/revoke at the account level. Permissions can be given at the user, group, or service principal level.

The best practices recommendation is to create a single Metastore for each of the workspaces' regions.

Instructions on creating Metastore in AWS and in Azure.

Enable your Workspaces

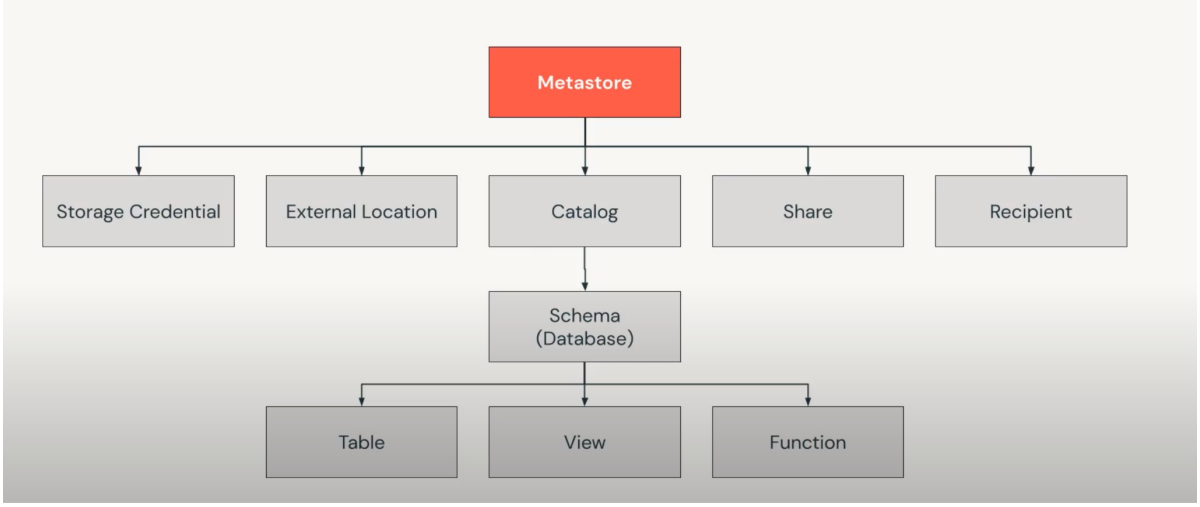
Unity Catalog is GA. You can continue to use the existing built-in Hive Metastore as your catalog which will show up as *hive_metastore* in your workspace. However, if you wish to take advantage of the new capabilities of Unity Catalog, you can attach a metastore and start moving your existing databases and tables over to the attached Unity Catalog metastore. You can continue to use *hive_metastore* or Glue Catalog alongside Unity Catalog metastores within a workspace without interruptions to your current processes as you go through the migration process.

Note: Even though UC enables managing access controls across workspaces, we still recommend limiting the number of workspaces to reduce administrative overhead.

Step 3 - Design your catalog layout

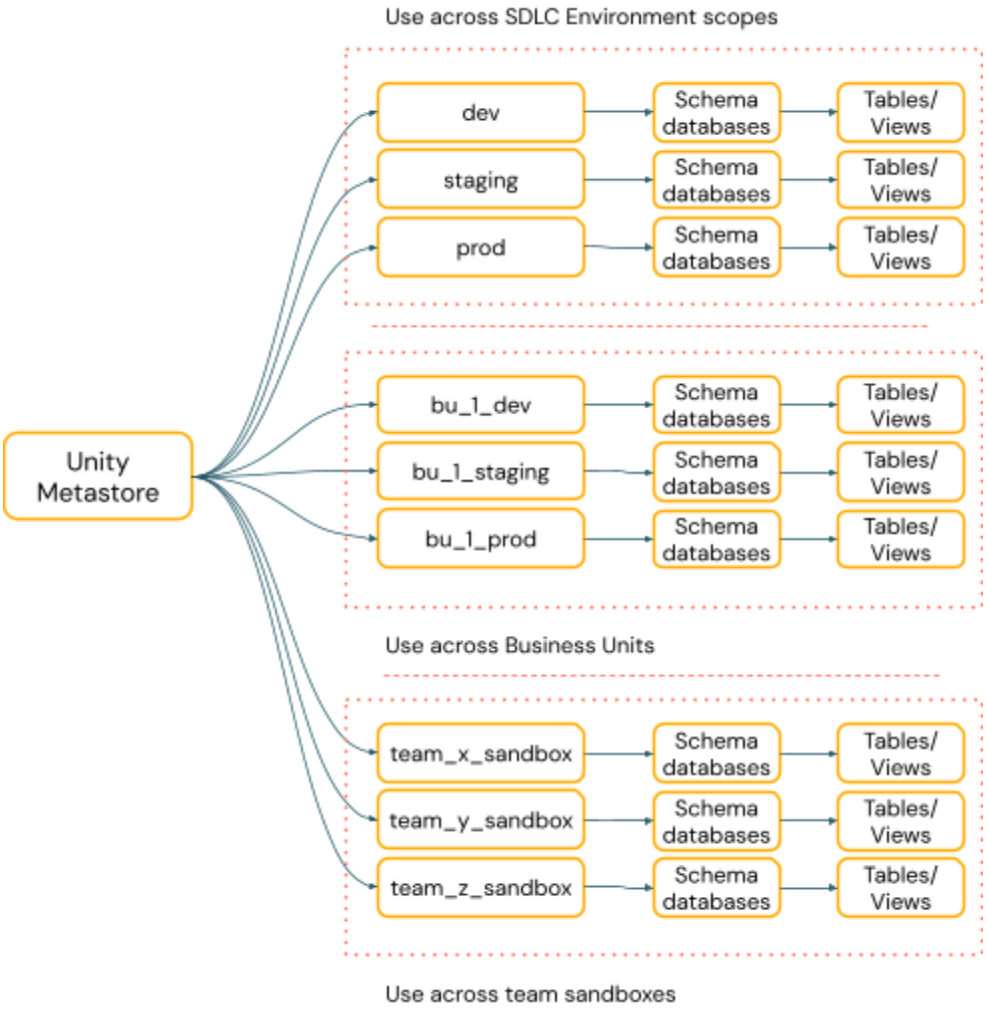
Designing Catalog Structure

Once the Metastore is created and assigned to a workspace, we can start designing and implementing the metadata layout. Migrating to Unity Catalog moves your users from a two level namespace in hive metastore to a three level namespace. Each catalog is made up of databases/schemas and each database (or schema) is made up of tables and views. See the picture below on what makes up the metastore.



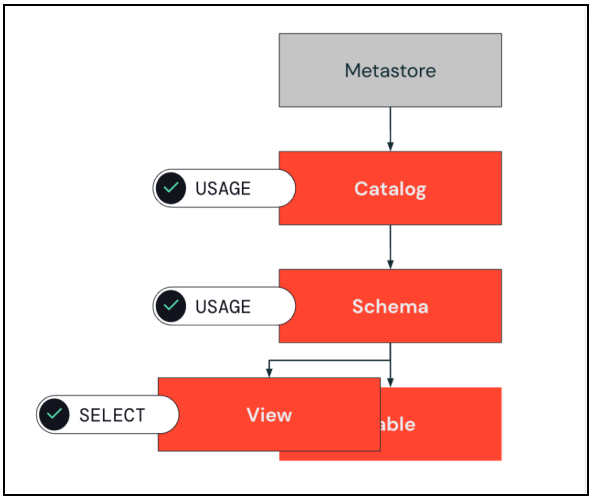
The first decision point is how to segregate the data by Catalogs. The catalog is a new data element in UC that groups a number of Databases/Schemas. Each Catalog can be assigned ACLs to govern who can access the catalog. Metastore admins and those with ownership privileges of a catalog are the persona responsible for creating the schemas.

Catalogs should be used to provide segregation across your organization's information architecture. Often this means that Catalogs can correspond to environment scope, team, or business unit, or some combination of two or three.



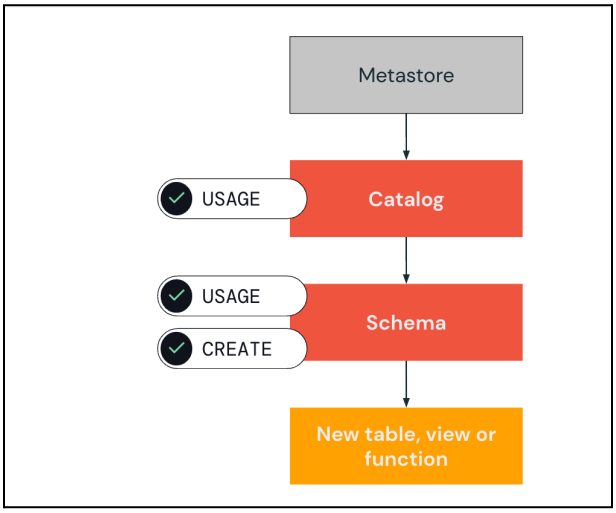
Permission granting items to remember when designing your information layout:

- Only allow MODIFY access to service principals. Individuals should not update production data manually.
- Remember security and visibility rules:
    - If you want a principal to see schemas in a catalog
        - GRANT USE CATALOG on <catalog> to <group>
    - If you want a principal to see tables/views in a schema
        - GRANT USE SCHEMA on schema to <group>
        - GRANT SELECT on <table/view> to <group>
- Permissions are not inherited so you have to grant the group USAGE on the catalog and the schema before granting SELECT on the table/view for the group. See the image below that illustrates this point.



Additionally, to give permission to create a new table, view or function, you need to grant the group the privileges for USAGE on the catalog and then the schema first. Lastly, you grant CREATE privileges on the schema for the group.

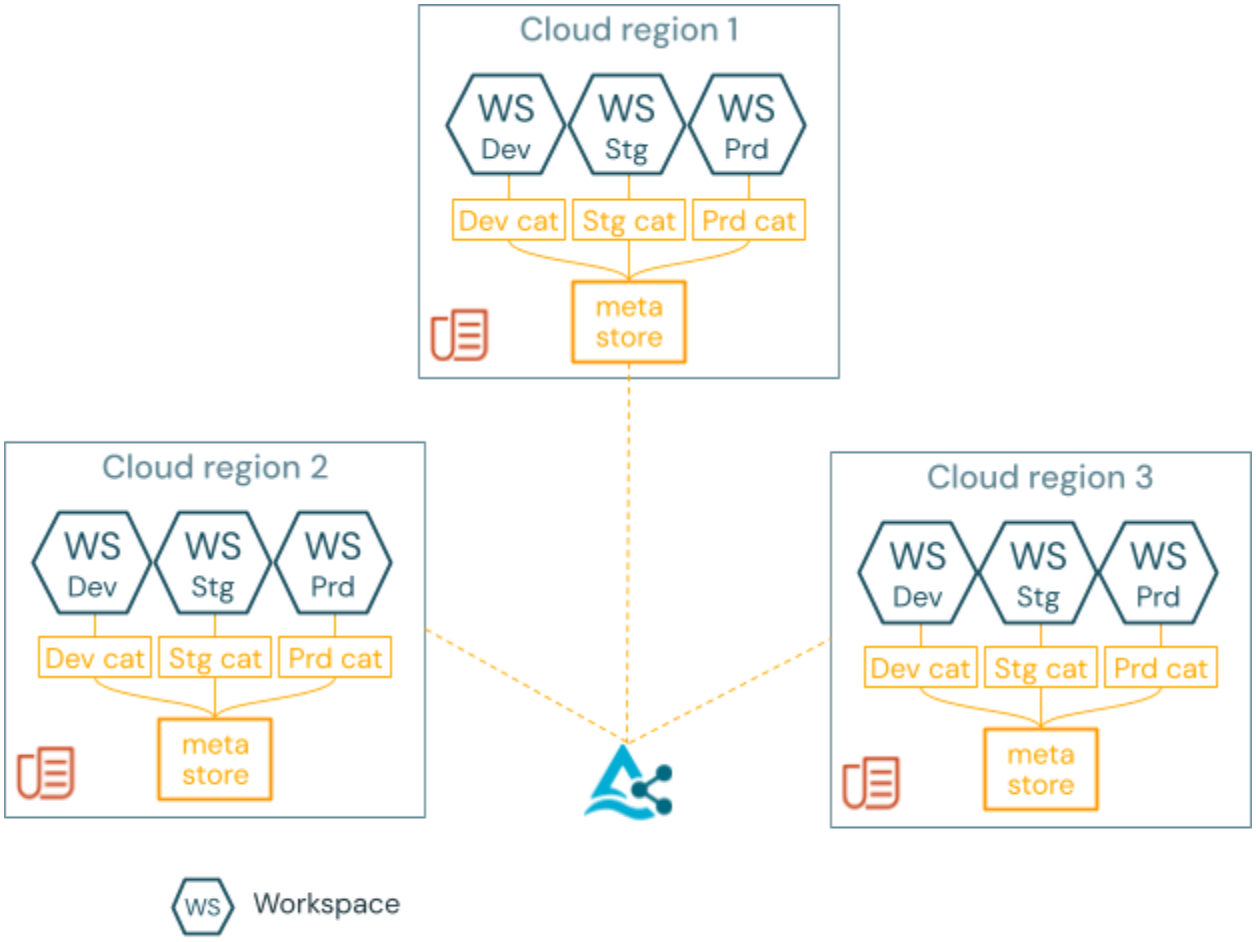
- GRANT USAGE on <catalog> to <group>
- GRANT USAGE on schema to <group>
- GRANT CREATE on schema to <group>



Best Practice

- Organize your Catalogs by Software Development Lifecycle environment scope, by business unit, or by both.
- Organize Catalogs additionally by teams for shared workspaces and sandboxes
- Always make the owner of Production Catalogs/Schemas groups and not individuals
- Only grant USAGE to Catalogs and Schemas when you want people to see and/or query the contents

Multi-region/cloud Metastores with Delta Sharing



Multi cloud/region metastores with Delta Sharing

It is not possible to use a metastore from region A in another region B. As Unity Catalog offers a low latency metadata layer, this would impact query performance and as such, it is not a supported or allowed configuration.

Metastore segregation occurs at the boundary point between cloud regions or cloud providers. This means that any data defined in one metastore that needs to be available in the other metastore will need to be shared.

To share data between metastores, customers should leverage **Databricks-managed Delta Sharing**, a.k.a. Databricks-to-Databricks Delta Sharing.

This allows you to register tables from metastores in different regions. These tables will appear as read-only catalogs/databases/tables in the consuming metastore. These tables can be granted access like any other object within Unity Catalog.

Keep in mind the following caveats:

- Lineage graphs are created at the metastore level, and will not cross Databricks to Databricks Delta Sharing. Even if a resource is shared intra-account (within the same Databricks Account) to another metastore, lineage information from the source will not be visible in the destination, and vice versa.
- ACLs are at the metastore level, and will not cross Databricks to Databricks Delta Sharing. If a resource has ACLs on it and that resource is shared to an intra-account metastore, then the ACLs from the source will not apply to the destination share. The destination share will have to set its own ACLs accordingly.

Registering External Tables Across Multiple Metastores

We strongly **do not recommend** registering common tables as External Tables in more than one metastore. There is a risk here in that any changes to the schema, table properties, and comments that occur as a result of writes to metastore A will not register at all with metastore B. Tables in metastore B would have to be recreated in order to have the correct schema, table properties and comments would be entirely disconnected otherwise. This can also cause consistency issues with the Delta Commit service. Use DB 2 DB Delta Sharing for sharing data between metastores.

Multi-region Best Practice

- Use your single region metastore for all software development life cycle scopes and business units, i.e. dev/test/prod/sales/marketing, etc.
- Use Databricks to Databricks Delta Sharing between cloud regions or cloud providers
    - Note when you share data, however, the data will be read only in the target workspace.
- Use Delta Shares for tables that are infrequently accessed, as you will be responsible for egress charges from cloud region to cloud region if you have frequently accessed data it may make sense to copy it across regions via a process and then query it as opposed to querying it across regions.

Step 4 - Upgrading your tables for Unity Catalog

It is good to note here that your UC catalogs can co-exist with hive_metastore catalogs and your current processes can continue uninterrupted with hive_metastore. However, you have to upgrade your tables if you want to take advantage of the UC capabilities. Therefore, make sure you do not fall into any of these UC GA Limits before you do the following steps. Once again, you can use your legacy hive_metastore with the UC catalogs but there are different security models between the two. Therefore, you can keep both during the table migration process but it is not suggested to keep both for a long time for the sake of in-sync security.

Upgrading Tables/Databases

Once the Catalogs are created we can migrate the existing data assets to Unity Catalog. Please note that in-place upgrade utility is only available for external tables where you specified an external S3 or ADLS Gen 2 storage container in the LOCATION path of a CREATE TABLE tablename command or by providing a path using saveAsTable in Python.

Upgrading In Place External Locations

Here are links explaining what External locations are and how to use them with Unity Catalog in AWS and Azure. **Note**: **A prerequisite for upgrading an external table is that an external location is created for the location. Additionally, upgrading external tables requires elevating service principals/instance profiles to the account level.**

The first step is identifying all the locations/Storage Accounts used to store external tables and create the necessary locations and credentials to support them. External Tables migration is simple and supported by the UI and by new synchronization commands. Here is a link to the instructions.

Migrating External Tables creates the metadata for an existing table in UC relying on the same underlying set of files. File formats supported for external location upgrade **only** are: TEXT, AVRO, CSV, JSON, JDBC, PARQUET, ORC, HIVE, DELTA, or LIBSVM. However, Hive metastores need to be upgraded separately with the instructions further in this section.

Best Practices

- Use the Upgrade Wizard to Upgrade External Tables
- Recreate view definitions after all of the referenced tables are upgraded

Managed Tables

Managed tables are saved in DBFS with Legacy Hive Metastores by default. File storage in DBFS is not supported by UC therefore all managed tables will have to be copied over to the UC storage account (the location is specified when you created your metastore). If the DBFS root bucket was used with UC to grant privileges on managed tables, this could lead to security issues of the entire root bucket. Therefore, the managed tables must migrate to a separate location to ensure account level security is maintained. **Note: If your managed table location was set at the database level with the hive metastore outside of DBFS then follow external location instructions above. Additionally, the only data format that can migrate is data in the delta format.**

Here is a link to instructions on how to migrate your managed tables for Unity Catalog. Deep Clone can be used to copy data from legacy managed tables to Unity Catalog Managed tables if the tables are in delta format. Here is a script for assisting the AWS managed table upgrade.

For managed storage locations, follow the general guidance below:

- Do not use any bucket or container that another service/group/user has access to.
    - This will allow those services/groups/users to bypass the managed storage location in Unity Catalog, breaking access control and audit-ability on managed tables.
- Do not reuse a bucket or container which was previously or is your current DBFS root file system for your managed storage location.
    - For the same reasons as above.
- If you set up an external location, database, or table in Unity Catalog then do not use the legacy access controls on those Unity Catalog assets.
- Note: Data is secure by default. You have to grant access to the catalog, database/schema, and table for a user to gain access to that data item.

Best Practice

- Manually CLONE/CREATE TABLE AS after identifying individual managed tables
    - Consider the size of tables and how long these tables will take to copy
- Recreate view definitions after all of the referenced tables are upgraded
- In the future, there will be benefits for having managed tables in Unity Catalog such as auto-tuning. Therefore, consider building up your data in the Unity Catalog managed tables.

Update Hive metastore data stored externally

Upgrade Overview

There are a few variants that you would have deployed HMS objects.

- In Spark, a table can either be MANAGED or EXTERNAL
- The location of the data can either be on Databricks File Storage (DBFS), or outside, i.e on a different cloud storage

This gives us 5 potential scenarios

1. Managed tables on DBFS - this means the files reside completely within DBFS and the only way forward for these hive metastore (HMS) objects is to recreate these tables via CLONE or CTAS
2. Managed tables outside of DBFS - this is when the parent database has its location set to external paths, e.g. a mounted path from the object store
3. External tables on DBFS - this is very rare, but prevalent, e.g. when someone does CREATE TABLE student (id INT, name STRING, age INT) LOCATION 'dbfs:/temp/student';
4. External tables outside of DBFS - this is supported by our upgrade utility
5. External tables using mount points

These below instructions outlines ***scenario 2 and 4 only for now***

Upgrade Challenges

The two main challenges with migrating HMS objects for customers that predominantly use managed tables only:

- The managed tables reside wholly within DBFS, and so migrating these objects from HMS to UC will also involve physical migration of data files or employ cloning of delta tables. This becomes a bigger challenge for large tables
- Even if the managed tables use file paths that are mounted from the object storage, and these mount points can be migrated to UC external locations, there is still an ongoing risk of someone accidentally dropping the managed table in HMS thereby deleting all the data files with it. Also, using this approach, the entitlements will have to be maintained effectively in 2 places (managed tables in HMS and external tables in UC) which becomes problematic to manage at scale

Upgrade Instructions

Prerequisites

Whilst the different upgrade paths outline the steps to migrate existing HMS table objects to UC, it is imperative that suitable access mechanisms be in place (instance profiles/Service principals for existing HMS objects and any IAM roles/SPs/MIs with the right access privileges in the target bucket/containers where the UC external tables will be instantiated.

Scenario 2: Using managed tables on HMS with mounted file paths

The following steps outline the activities that need to be carried out to safely migrate existing HMS *managed* tables with mount based file paths to UC external tables

Step 1: Finalize the list of prospective candidate HMS table objects for the proposed upgrade.

Step 2: We will proceed to check the table status to ensure that the selected table objects will fit this pattern of upgrade

```
1  %sql
2  DESCRIBE TABLE EXTENDED ad_intel.ad_type
```

Table +

	col_name	data_type	comment
17	Location	dbfs:/user/hive/warehouse/ad_intel.db/ad_type	
18	Provider	delta	
19	Owner	root	
20	Is_managed_location	true	
21	Type	MANAGED	
22	Table Properties	[delta.minReaderVersion=1,delta.minWriterVersion=2]	

Showing all 22 rows. | 0.40 seconds runtime

Step 3: Run a script that converts the managed to an external table by accessing the Spark catalog

```
import org.apache.spark.sql.catalyst.catalog.{CatalogTable, CatalogTableType}
import org.apache.spark.sql.catalyst.TableIdentifier

val tableName = "ad_type"
val dbName = "ad_intel"

val oldTable: CatalogTable =
spark.sessionState.catalog.getTableMetadata(TableIdentifier(tableName, Some(dbName)))
val alteredTable: CatalogTable = oldTable.copy(tableType = CatalogTableType.EXTERNAL)
spark.sessionState.catalog.alterTable(alteredTable)
```

Step 4: Now check the status of the table to verify if this change has taken effect

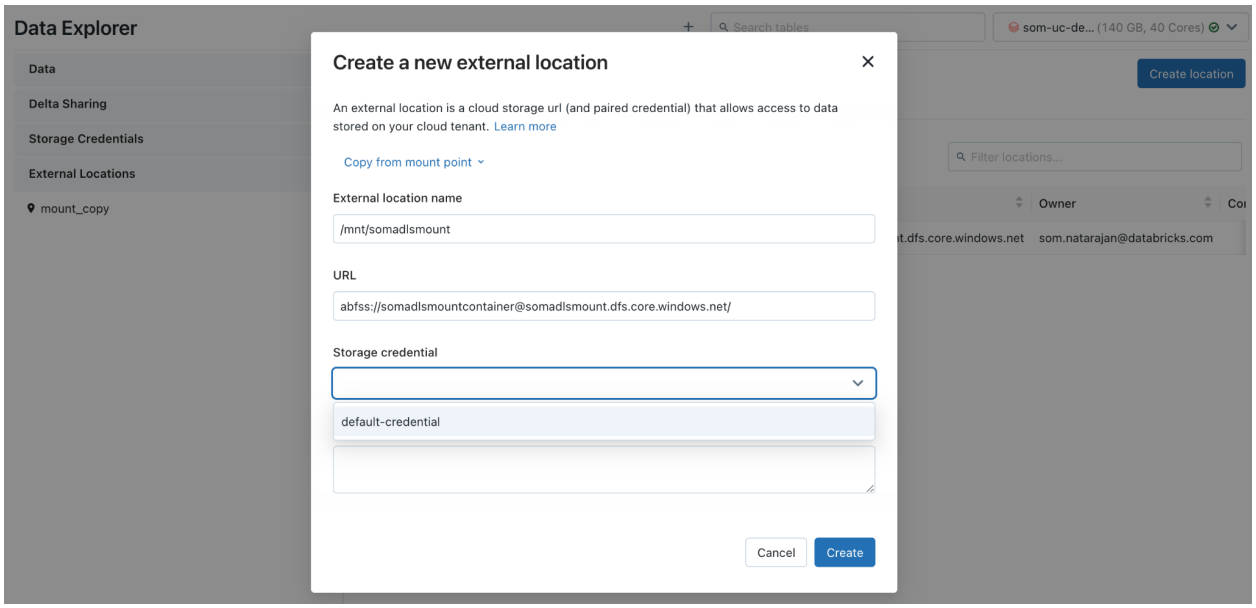
```
%sql
DESCRIBE TABLE EXTENDED ad_intel.ad_type
```

Table

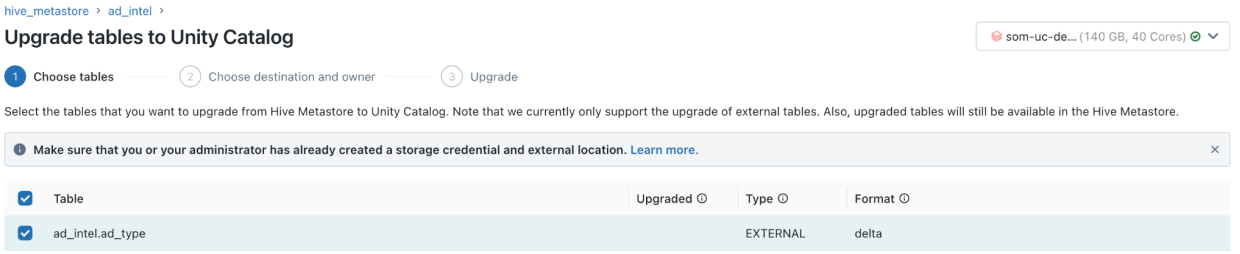
	col_name	data_type	comment
17	Location	dbfs:/user/hive/warehouse/ad_intel.db/ad_type	
18	Provider	delta	
19	Owner	root	
20	External	true	
21	Type	EXTERNAL	
22	Table Properties	[delta.minReaderVersion=1,delta.minWriterVersion=2]	

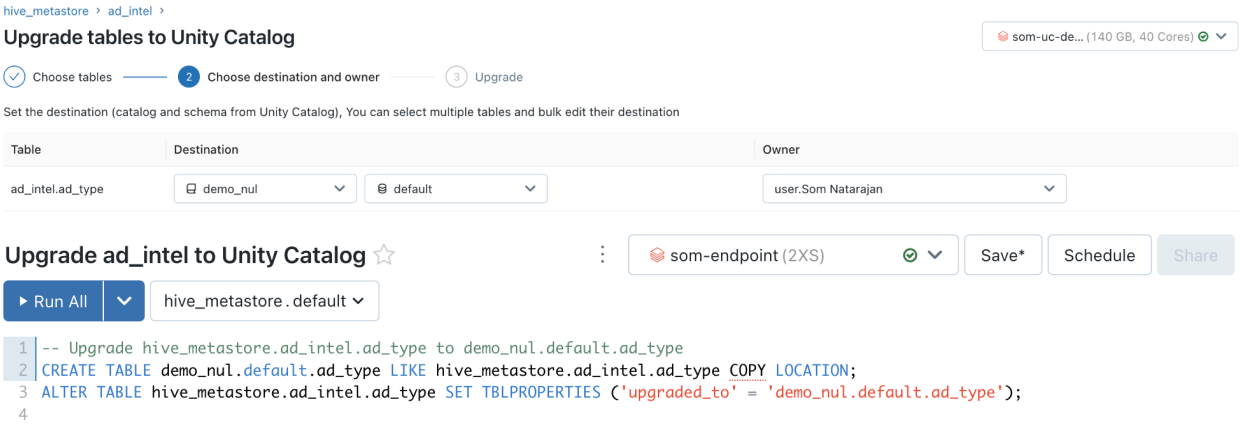
Showing all 22 rows. | 0.52 seconds runtime

Step 5: Create UC external locations for each mount point location for the tables selected for this upgrade either in the same script outlined in Step 3 or via a separate procedure/UI (The UI path is depicted below) using the preferred *storage credential*



Step 6: Add the final step of creating an external table in UC in the same script or this action can be performed separately. However, scripting it all in one will help with bulk upgrading lots of table objects (The UI path is depicted below)





Repeat the above steps for all the other candidate HMS objects if performing these from the UI, or script them into the master script and loop them over to perform these actions together. *Please note that the UI upgrade for external HMS tables is only available via DBSQL Warehouse endpoints.*

Scenario 4: External tables outside of DBFS

The following steps outline the activities that need to be carried out to safely migrate existing external tables residing outside of DBFS to Unity catalog external tables.

Step 1: Finalize the list of prospective candidate HMS table objects for the proposed upgrade.

Step 2: We will proceed to check the table status to ensure that the selected table objects will fit this pattern of upgrade

Step 5 - Setting up your data access controls with Unity Catalog

Upgrading Data Access Controls

By default every data object created in UC can be accessed only by the Owner and the Metastore Admin. Any other users will be blocked by default. Once the tables are migrated, privileges have to be granted to individual users and groups. Access to the underlying storage is performed using the credentials provided to the "External Location". Credentials and External Location permissions are overridden by the Table Permissions.

Instructions for managing privileges in Unity Catalog are here. Best practices can be found on this Databricks Community post.

Setting Up Access Control

To recap from a previous section, in order to read data from a table or view, a user must have the following:

- SELECT on the table or view
- USAGE on the schema that owns the table
- USAGE on the catalog that owns the schema

Securing Access to Tables Only

```
GRANT USE CATALOG cat to `group_name`;
GRANT USE SCHEMA cat.schema to `group_name`;
GRANT SELECT on cat.schema.table to `group_name`;
```

Securing Access to Rows/Columns

Securing Access to Columns

- Use a dynamic view
- Use a secondary "view" schema to avoid polluting the primary schema.

```
CREATE VIEW cat.secured.table as
  SELECT
    id,
```

```
    CASE WHEN
      is_member('group_name') THEN email
      ELSE 'REDACTED'
    END AS email,
    country,
    product,
    total
  FROM cat.db.table


GRANT usage on catalog cat to `group_name`;
GRANT usage on database cat.secured to `group_name`;
GRANT select on cat.secured.table to `group_name`;
```

Securing Access to Rows

- Use a dynamic view
- Use a secondary "view" database to avoid polluting the primary database.

```
CREATE VIEW cat.secured.table as
  SELECT
    *
  FROM cat.schema.table
  WHERE
    CASE
      WHEN is_member('managers') THEN TRUE
      ELSE total <= 1000000
    END;


GRANT usage on catalog cat to `group_name`;
GRANT usage on schema cat.secured to `group_name`;
GRANT select on cat.secured.table to `group_name`;
```

Note that **users can only use User Isolation Clusters** when using dynamic views. **All the below qualifying statements** must also be true in order to provide appropriate security.

- User is granted SELECT permission on a view X_RESTRICTED which references table X
    - This view is owned by a user or principal that has SELECT access on table X.
- User does **not** have permissions to access data directly from X

Best Practice

- Secure access to tables via groups
- For column level selection, row filtering or masking, use dynamic views
- You must use User Isolation Clusters to use dynamic views.
- UC will support ACLs on tables & external storage locations for both R & Scala, but only in single user mode

Step 6 - Setting up your clusters, workloads, and downstream tools

Before you set up your clusters, note that there are different types and you should plan your new Unity Catalog clusters accordingly. See the image below and go to this link for further details.

Access Mode	Visible to user	UC Support	Supported Languages	Notes
Single User	Always	Yes	Python, SQL, Scala, R	Can be assigned to and used by a single user. To read from a view, you must have `SELECT` on all referenced tables and views. Dynamic views are not supported. Credential passthrough is not supported.
Shared	Always (**Premium plan required**)	Yes	Python (on Databricks Runtime 11.1 and above), SQL	Can be used by multiple users with data isolation among users. See shared limitations.
No Isolation Shared	Admins can hide this cluster type by enforcing user isolation in the admin settings page.	No	Python, SQL, Scala, R	There is a related account-level setting for No Isolation Shared clusters.
Custom	Hidden (For all new clusters)	No	Python, SQL, Scala, R	This option is shown only if you have existing clusters without a specified access mode.

Note that shared clusters can have third party libraries through %pip install - see here for how.

Setting up Clusters

Follow these instructions to enable UC on your clusters and SQL warehouse. Today, shared clusters come with a number of limitations: we only support Python and SQL and a number of features present on other cluster types are not supported, see full list. We are working on adding support for many of the missing features. Please sign up to be enrolled into our private preview.

Setting up Cluster Policies

Create cluster policies which restrict access to only create clusters which are UC and lineage enabled.

Using cluster policies reduces available choices, which will greatly simplify the cluster creation process for users and ensure that they are able to access data seamlessly.

Assign these policies to all users that have the Create cluster right to ensure consistent usage of UC enabled clusters.

Single-user cluster policy

```
{
  "spark_version": {
    "type": "regex",
    "pattern": "1[0-1]\\.[0-9].*",
    "defaultValue": "10.4.x-scala2.12"
  },
  "data_security_mode": {
    "type": "fixed",
    "value": "SINGLE_USER",
    "hidden": true
  },
  "single_user_name": {
    "type": "regex",
    "pattern": ".*",
    "hidden": true
  },
  "spark_conf.spark.databricks.dataLineage.enabled": {
    "type": "fixed",
    "value": "true"
  }
}
```

If using Glue/external metastores, then you need to add the appropriate Spark configurations to the above cluster policy, e.g.

```
{
  "spark_version": {
    "type": "regex",
    "pattern": "1[0-1]\\.[0-9].*",
    "defaultValue": "10.4.x-scala2.12"
  },
  "data_security_mode": {
    "type": "fixed",
    "value": "SINGLE_USER",
    "hidden": true
  },
```

```
  "single_user_name": {
    "type": "regex",
    "pattern": ".*",
    "hidden": true
  },
  "spark_conf.spark.databricks.dataLineage.enabled": {
    "type": "fixed",
    "value": "true"
  }
  "spark_conf.spark.databricks.hive.metastore.glueCatalog.enabled": {
    "type": "fixed",
    "value": "true"
  }
}
```

Setting up the user isolation cluster policy

Note that the below policy enables Multi User Python on User Isolation Clusters. This is the recommended best practice.

```
{
  "spark_version": {
    "type": "regex",
    "pattern": "1[0-1]\\.[0-9]*\\.x-scala.*",
    "defaultValue": "10.4.x-scala2.12"
  },
  "data_security_mode": {
    "type": "fixed",
    "value": "USER_ISOLATION",
    "hidden": true
  },
  "spark_conf.spark.databricks.dataLineage.enabled": {
    "type": "fixed",
    "value": "true"
  }
}
```

Upgrading Clusters to UC

Upgrade Scenarios:

Here are a few scenarios to look for in order to upgrade

1. Cluster running DBR less than 11.1
2. Cluster running DBR 11.1 and above
    a. Interactive cluster
    b. Automated cluster

Limitations:

- Scala, R, and workloads using the Machine Learning Runtime are supported only on clusters using the single user access mode. Workloads in these languages do not support the use of dynamic views for row-level or column-level security.
- In the Advanced options section, the credential passthrough checkbox is available when the access mode is Single user or Shared. When you select the credential passthrough checkbox, the cluster supports passthrough and Unity Catalog access is disabled.
- The following security-related Spark configurations are not available in the new UI when creating a new cluster. If you need to fine-tune a cluster configuration (for example, for integrations such as Privacera or Immuta) you can define a custom cluster policy to set these configurations.

  Contact your Databricks representative for assistance.

  1. spark.databricks.repl.allowedLanguages
  2. spark.databricks.acl.dfAclsEnabled
  3. spark.databricks.passthrough.enabled
  4. spark.databricks.pyspark.enableProcessIsolation"

Prerequisites

- UC should be enabled for the account
- At least one metastore has to be created and assigned to the workspace
- Catalogs/Databases/Tables have to be created and assigned proper ACLs

Upgrade Instructions:

Scenario 1: Clusters running DBR less than 11.1

With this scenario you will have to tear down the cluster and rebuild it, there's no way to upgrade clusters to use UC for this scenario. Follow instructions in scenario two (2) to get clusters to UC

Scenario 2: Clusters running DBR 11.1+

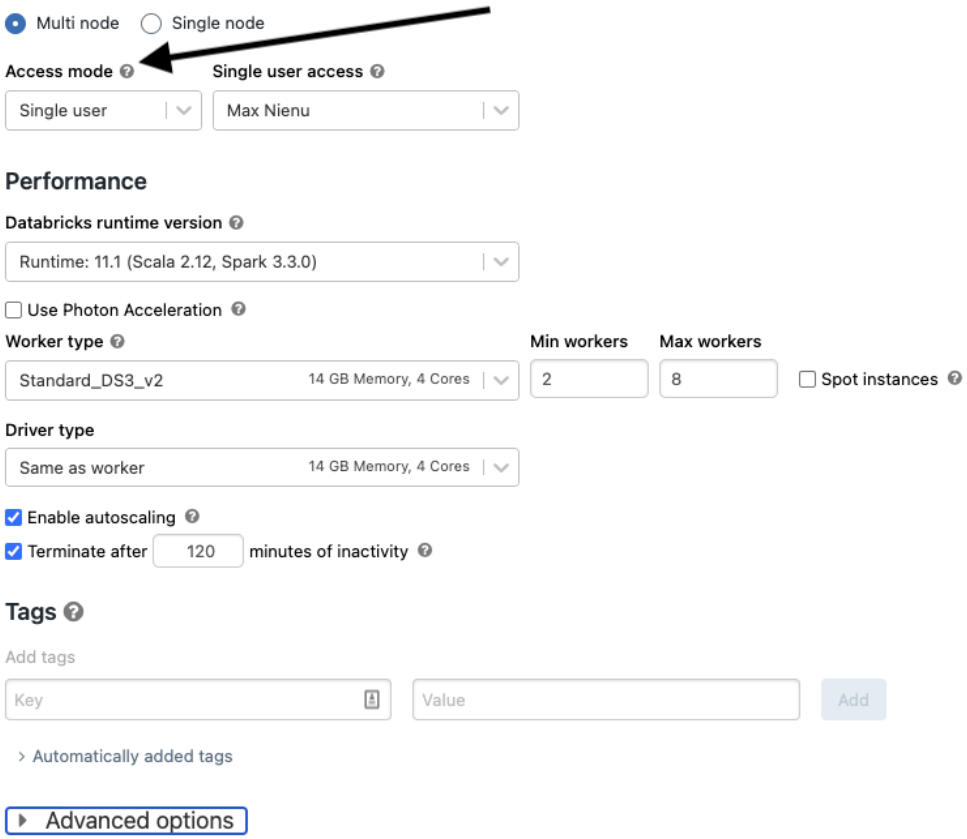
With this scenario you can choose the different access modes for the cluster to be used. There are three (3) access modes here

- **Single user:** This mode allows you to Run SQL, Python, R and Scala workloads as a single user, with access to data secured in Unity Catalog.
- **Shared:** Multiple users can share the cluster to run SQL and Python workloads on data secured in Unity Catalog.

- **No isolation shared:** Multiple users can share the cluster to run workloads in any language, with no isolation between users.

The first two (2) access modes leverage UC, to upgrade from the no isolation shared mode to the UC leveraged security mode, edit your cluster and change the access mode to either the single user or shared from the access mode dropdown

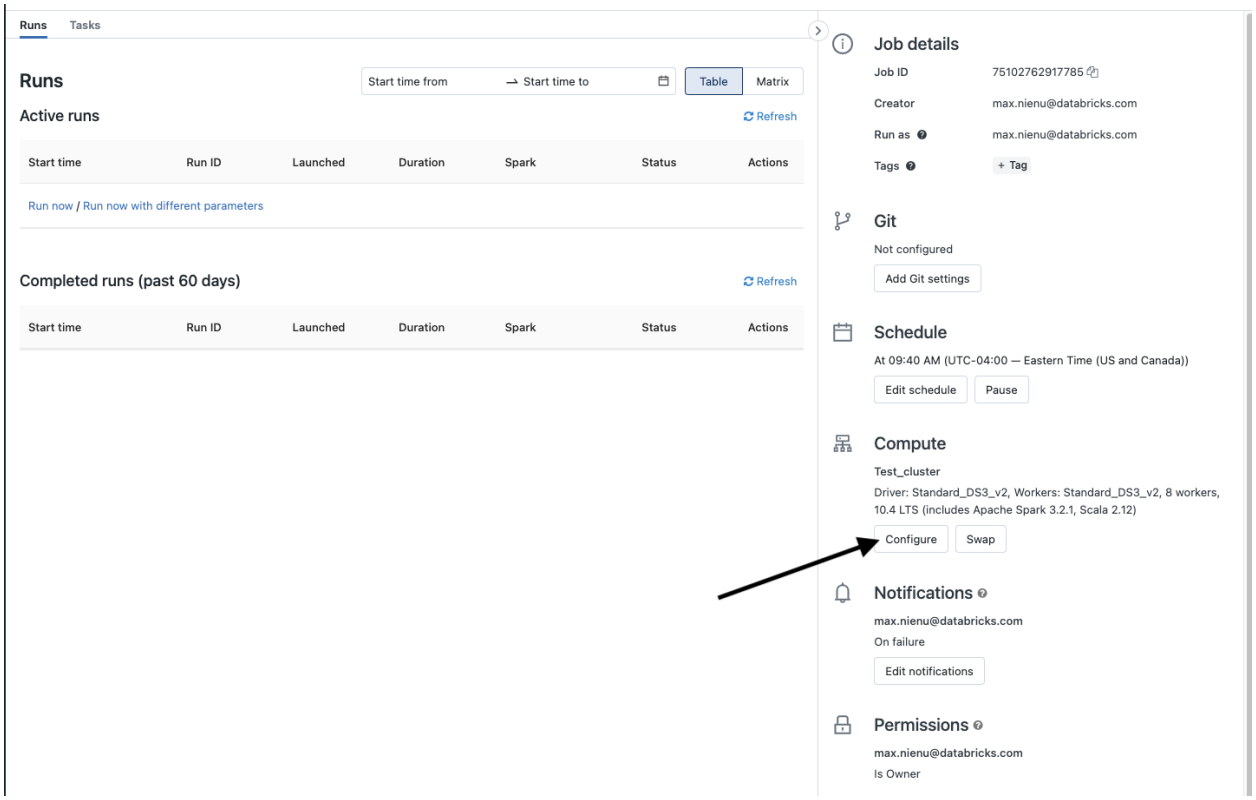
Interactive Cluster:
To switch an interactive cluster to UC mode, click on the Edit button on the cluster page and click on the access mode and change to either Single user or shared to leverage UC



To add default catalog, you can use the config below in the spark settings:

```
spark.databricks.sql.initial.catalog.name name_of_catalog
```

Automated Cluster:
Switching automated clusters to use UC follows the same prerequisites for interactive clusters. To do this, go to the job details screen and click configure under Compute



That should bring up the cluster configuration screen and you can click on either Single User or Shared in the access mode drop down to leverage UC.

Upgrading Workflows

Once the data is migrated to Unity Catalog the existing jobs will have to be updated to Unity Catalog.

The first step is modifying the job cluster to support UC which was shown above. The second step is to either (1) change your code to use the three level namespace **or** (2) to define a default catalog on the cluster.

To set a default catalog is performed by setting the following Spark Configuration option
`spark.databricks.sql.initial.catalog.name <catalog name>`

See below for more in depth instructions to help upgrade your workflows.

Note: **Extensive testing should be done to assure that the new DBR version and three level namespace didn't break any existing functionality.** Before you drop the old table, test for dependencies by revoking access to it and re-running related queries and workloads. Also check the UC GA limits section of the release notes to make sure the functionality is possible with UC.

Upgrade Scenarios

We will look into a few scenarios that require an upgrade.

1. Using a managed table as a target for batch
2. Using an external table as a target for batch

Upgrade Challenges

The main challenges with upgrading batch workloads are:

a. UC doesn't support DBFS. Any data (managed table currently stored in DBFS will have to be moved to the Client's cloud storage account. S3 in AWS and ADLS in Azure.
b. UC uses "External Locations" to write to the cloud storage account. "External Locations" block access to folders where tables have been created.
c. While changing jobs to point to the new UC tables, it helps if the database names aren't changed. You can just add 'USE <catalog name>' at the top of the job and the rest of the query doesn't change, or set the default catalog via the cluster config `spark.databricks.sql.initial.catalog.name <catalog-name>` . However if you change database names as well, you will have to update every query to change <database>.<table> to <new catalog>.<new database>.<table>.

Prerequisites

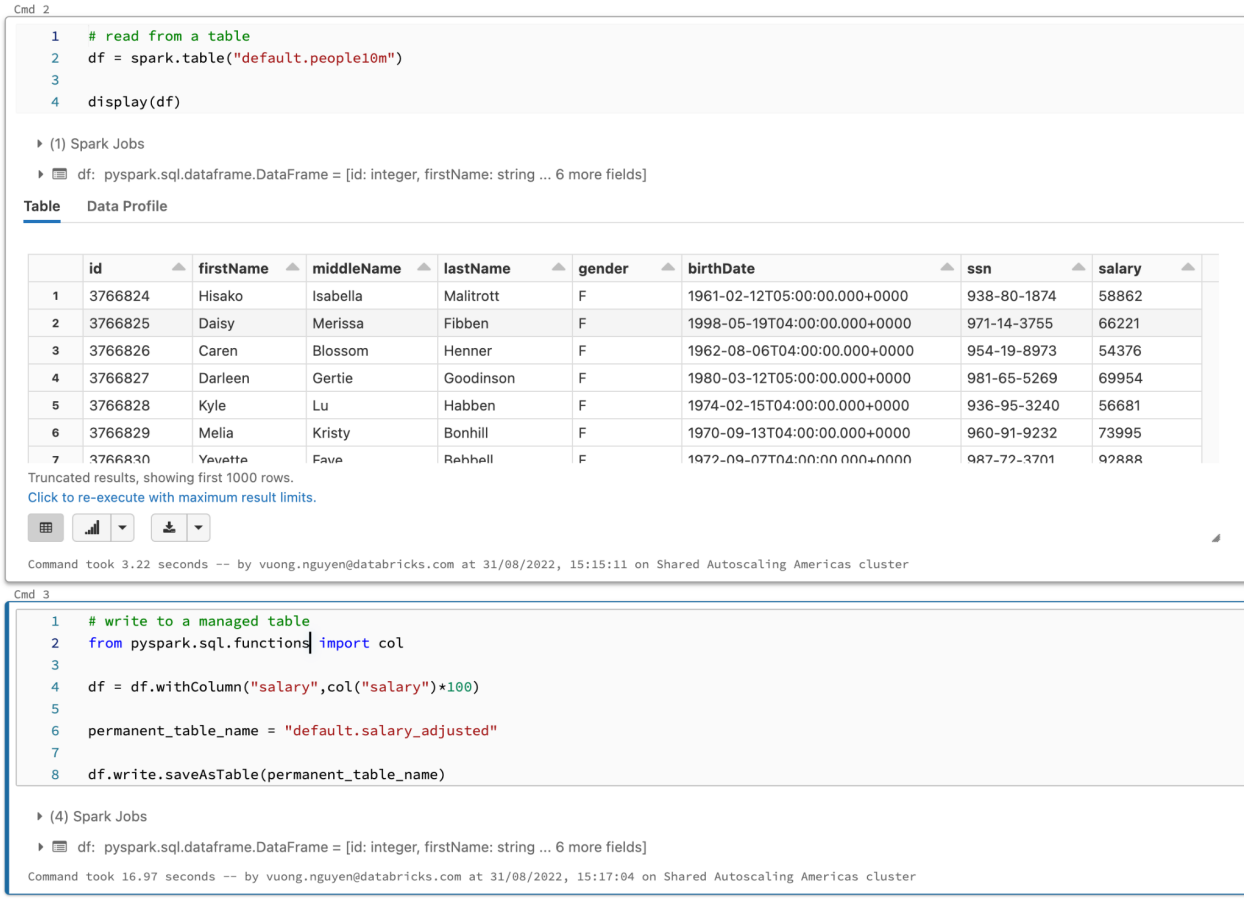
- UC should be enabled for the account.
- Metastore has to be created and assigned to the workspace.
- Catalogs/Databases/Tables have to be created and assigned the proper ACLs.
- A UC enabled cluster has to be created. UC GA is supported on DBR 11.1 or later.
- The Managed Tables examples apply only to Delta Tables.

Upgrade Instructions

Scenario 1: Using a managed table as a target for batch

With this scenario we use a managed table as a target for batch jobs.
As part of the upgrade, we will copy the table content to an external location (from DBFS),

For this upgrade we will use the following example:



- This is a batch job that reads from a table, performs some transformation and saves to a managed table.

Upgrade Steps:

1. The following steps will be performed by the **Metastore Admin**.
2. Create a Unity Catalog enabled cluster (DBR 11.1 or later, "Single-User" Security Mode).
3. Create CREDENTIAL via the Data Explorer UI in DBSQL documentation
4. Create EXTERNAL LOCATION, either via the Data Explorer UI in DBSQL or with the below command documentation
5. Grant Permission to CREATE TABLES, READ FILES, and WRITE FILES to the user who will run the streaming job and the user performing the upgrade. It is recommended that a principal will run the job rather than a user.
6. Create a Catalog and Database for the target table.

Cmd 4

```
%sql
CREATE CATALOG uc_batch;
CREATE SCHEMA uc_batch.upgraded _sqldf
```

▸ _sqldf: pyspark.sql.dataframe.DataFrame

OK

SQL cell result stored as PySpark data frame `_sqldf` . Learn more

Command took 4.07 seconds -- by vuong.nguyen@databricks.com at 31/08/2022, 15:22:55 on Vuong Nguyen's Cluster

7. Grant access to the user performing the upgrade as well as to the user/principal running the streaming job.

Cmd 5

```
%sql
-- Grant Permissions to the People performing the Migration
GRANT USAGE ON CATALOG uc_batch TO `data_eng`;
GRANT USAGE, CREATE ON SCHEMA uc_batch.upgraded TO `data_eng`;
```

Cmd 6

```
%sql
-- Grant Permissions to the sp running the batch job
GRANT USAGE ON CATALOG uc_batch TO `batch_spn`;
GRANT USAGE, CREATE ON SCHEMA uc_batch.upgraded TO `batch_spn`;
```

8. The following steps can be performed by a developer. The developer has to be granted CREATE TABLES, READ FILES, and WRITE FILES rights on the external location. The users have to be granted access to a UC Database or permission to create one.
9. Stop the batch job. To assure a seamless transition to the new target, we will have to stop the job.
10. Deep clone the old target table to a newly created table. The new table can be a managed table or an external table. In both cases the table will be saved to the customer's storage account.

Cmd 7

```
%sql
-- Clone the target table to UC
CREATE TABLE IF NOT EXISTS uc_batch.upgraded.salary_adjusted DEEP CLONE hive_metastore.default.salary_adjusted _sqldf
```

▸ (10) Spark Jobs

▸ _sqldf: pyspark.sql.dataframe.DataFrame = [source_table_size: long, source_num_of_files: long ... 4 more fields]

Table | Data Profile

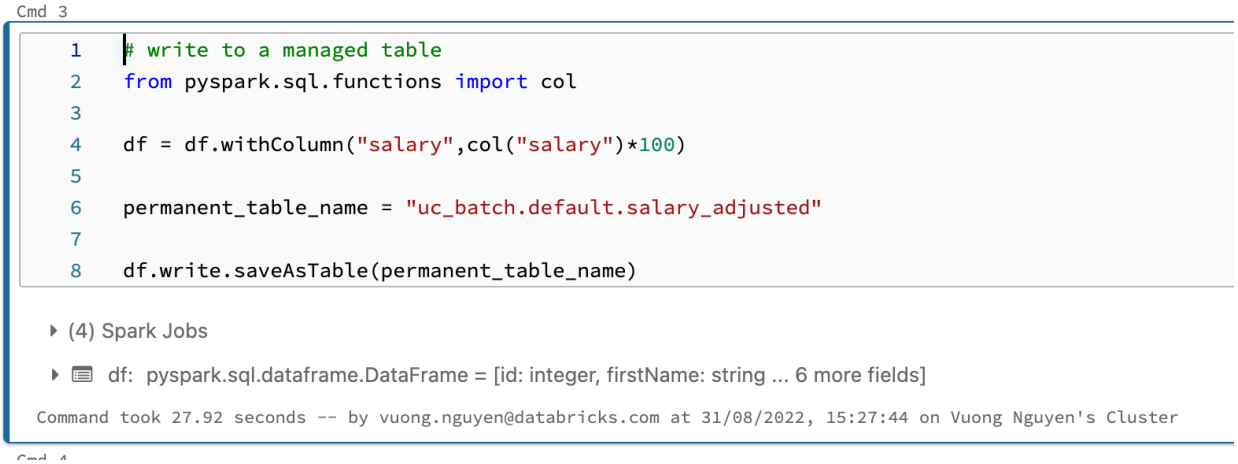
	source_table_size	source_num_of_files	num_removed_files	num_copied_files	removed_files_size	copied_files_size
1	236921471	8	0	8	0	236921471

Showing all 1 rows.

SQL cell result stored as PySpark data frame `_sqldf` . Learn more

Command took 40.16 seconds -- by vuong.nguyen@databricks.com at 31/08/2022, 15:26:11 on Vuong Nguyen's Cluster

11. Deprecate the old table. There are few options for deprecating the old tables. You can rename the table. Remove all the references to the old table.
12. Refactor the old batch code, i.e. point the toTable target to the newly cloned table.



13. Attempt to run the code interactively.
14. Create a Job with a UC enabled Job Cluster. Set the Security Mode to "Single-User" and assign a principal to the Job.

Potential Shortcuts

1. Mass append spark.sql("USE CATALOG <this-is-the-future>") to all existing codes
2. You can set the default catalog using the spark config as shown below
   `spark.databricks.sql.initial.catalog.name <catalog-name>`
3. If a majority of your queries in a workspace point to the same catalog then you can make that catalog the default catalog for the workspace using CLI commands as shown below.

```
databricks unity-catalog metastores assign —workspace-id 1234567890123456 \
                                          —metastore-id 7890-1cd2-3456-e789f0a12b34 \
                                          —default-catalog-name sales_catalog
```

Use Service Principals

Run jobs as Service Principals. Users can create jobs but should not run them, Workspace Administrators (dev ops) should assign a service principal with the requisite access required to complete the job. The service principal should be the Owner of the job. This operation can currently only be performed by a Workspace Administrator.

Job Management

Workspace Administrators can change job ownership, which means that they can by extension access data entitled to any service principal through Unity Catalog, *but only if that service principal is assigned to their workspace.*

It is recommended that the role of Workspace Administrator is tightly controlled, and that access is limited to required dev ops or IT ops groups.

Job Best Practice

- Use Service Principals
    - If jobs are created by users they should be assigned to be owned by Service Principals
- Lock down access to Workspace Administrator roles.

Upgrading your existing downstream tools for Unity Catalog

This section contains upgrade instructions for autoloader/streaming, DBSQL, ML, Fivetran, and DBT.

Upgrade Autoloader/Streaming to Unity Catalog

Upgrade Scenarios

We will look into a few scenarios that require an upgrade.

1. Using a managed table as a target for streaming/Auto Loader
2. Using an external table as a target for streaming/Auto Loader
3. Using a managed table as a source for streaming

Upgrade Challenges

The two main challenges with upgrading streaming workloads are:

a. UC doesn't support DBFS. Any data (managed table) and supporting data (Checkpoints and Schema) currently stored in DBFS will have to be moved to the Client's cloud storage account. S3 in AWS and ADLS in Azure.
b. UC uses "External Locations" to write to the cloud storage account. "External Locations" block access to folders where tables have been created.
c. We will not be able to save the checkpoint and Schema to the same location as the table.

Prerequisites

- UC should be enabled for the account.
- Metastore has to be created and assigned to the workspace.

- Catalogs/Databases/Tables have to be created and assigned the proper ACLs.
- A UC enabled cluster has to be created. Streaming with UC is supported on DBR11.1 or later.
- The Managed Tables examples apply only to Delta Tables.

Upgrade Instructions

Scenario 1: Using a managed table as a target for streaming/Auto Loader

With this scenario we use a managed table as a target for Auto Loader or other streaming sources. As part of the upgrade, we will copy the table content to an external location (from DBFS), we will copy the checkpoint locations to a specified external location and we will resume streaming.

For this upgrade we will use the following example:

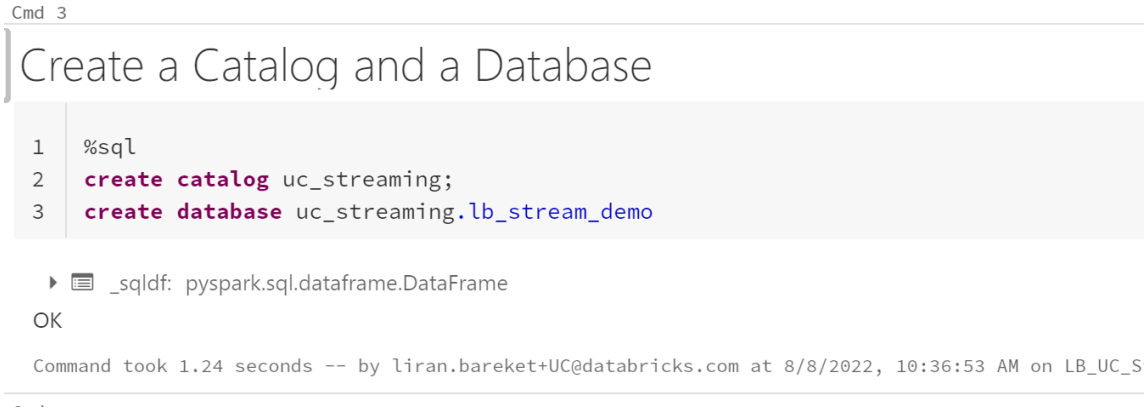
```
checkpoint_path = '/tmp/lb_streaming/_checkpoints'
schema_path = '/tmp/lb_streaming/_schema'
# checkpoint_path = 's3://databricks-ifi/sales-bucket/Autoloader/_checkpoints'
upload_path = 's3://databricks-ifi/sales-bucket/LB_AutoLoader_Input'

# Set up the stream to begin reading incoming files from the
# upload_path location.
df = spark.readStream.format('cloudFiles') \
  .option('cloudFiles.format', 'csv') \
  .option('header', 'true') \
  .option('cloudFiles.schemaLocation', schema_path) \
  .schema('city string, year int, population long') \
  .load(upload_path)

# Start the stream.
# Use the checkpoint_path location to keep a record of all files that
# have already been uploaded to the upload_path location.
# For those that have been uploaded since the last check,
# write the newly-uploaded files' data to the write_path location.
df.writeStream.format('delta') \
  .option('checkpointLocation', checkpoint_path) \
  .option("mergeSchema","true") \
  .toTable('lb_stream_demo.stream_pop')
```

- This is an Auto Loader example that reads csv files and saves to a managed table.
- In this example we are writing the stream using the "toTable" option. Another popular option is using the `.start(<write_path>)` notation.
- We specify the checkpoint location as a dbfs location, we specify a schema location as well.

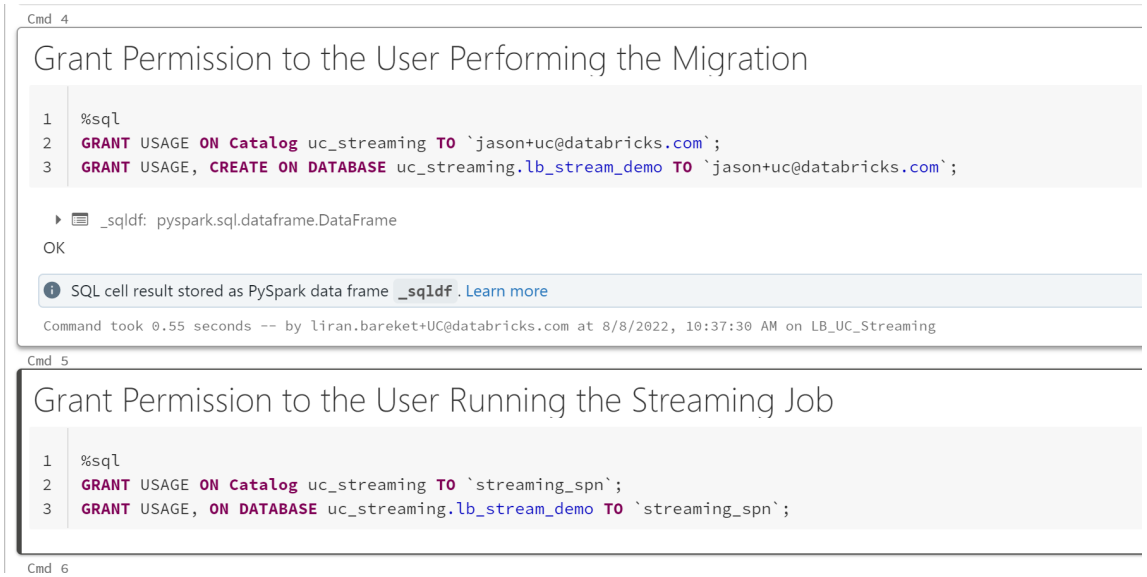
Upgrade Steps:

1. The following steps will be performed by the **Metastore Admin**.
2. Create a Unity Catalog enabled cluster (DBR 11.1 or later, "Single-User" Security Mode).
3. Create CREDENTIAL via the Data Explorer UI in DBSQL documentation

4. Create EXTERNAL LOCATION, either via the Data Explorer UI in DBSQL or with the below command [documentation]
5. Grant Permission to CREATE TABLES, READ FILES, and WRITE FILES to the user who will run the streaming job and the user performing the upgrade. It is recommended that a principal will run the job rather than a user.
6. Create a Catalog and Database for the target table (or use the ones you created during the "Designing Your Catalog Layout" step).



7. Grant access to the user performing the upgrade as well as to the user/principal running the streaming job.



8. The following steps can be performed by a developer. The developer has to be granted CREATE TABLES, READ FILES, and WRITE FILES rights on the external location. The users have to be granted access to a UC Database or permission to create one.
9. Stop the stream. To assure a seamless transition to the new target, we will have to stop the stream.
10. Copy the checkpointlocation to a new location on your cloud storage account. The new location should be under an "External Location" to get access protection. See

[Documentation]

Cmd 1

Copy Bookmark Location

```
src = "/tmp/lb_streaming/_checkpoints"
tgt = "s3://databricks-ifi/sales-bucket/Autoloader/_checkpoints/"
dbutils.fs.cp(src,tgt,True)
```

```
Out[1]: True
```

Command took 8.02 seconds -- by liran.bareket+UC@databricks.com at 8/8/2022, 10:26:52 AM on LB_UC_Streaming

Cmd 2

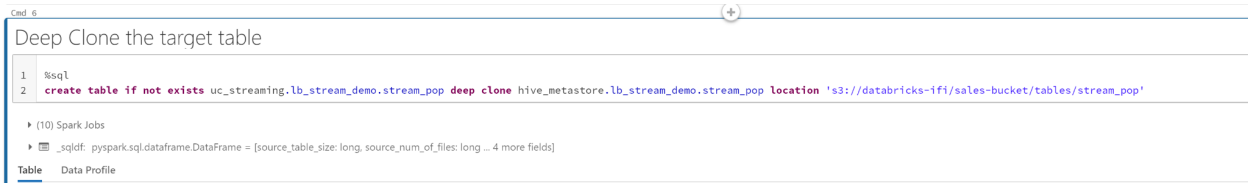
Copy Schema Location

```
src = "/tmp/lb_streaming/_schema"
tgt = "s3://databricks-ifi/sales-bucket/Autoloader/_schema/"
dbutils.fs.cp(src,tgt,True)
```

```
Out[2]: True
```

Command took 2.48 seconds -- by liran.bareket+UC@databricks.com at 8/8/2022, 10:26:57 AM on LB_UC_Streaming

11. Deep clone the old target table to a newly created table. The new table can be a managed table or an external table. In both cases the table will be saved to the customer's storage account.



12. Deprecate the old table. There are few options for deprecating the old tables. You can rename the table. Remove all the references to the old table.
13. Refactor the old streaming code. Apply the following change to the streaming code:
    a. Point the checkpoint path to the newly created path
    b. Point the schema path to the newly created path
    c. Point the toTable target to the newly cloned table.

```
checkpoint_path = 's3://databricks-ifi/sales-bucket/Autoloader/_checkpoints/'
schema_path = 's3://databricks-ifi/sales-bucket/Autoloader/_schema/'
# checkpoint_path = 's3://databricks-ifi/sales-bucket/Autoloader/_checkpoints'
upload_path = 's3://databricks-ifi/sales-bucket/LB_AutoLoader_Input'

# Set up the stream to begin reading incoming files from the
# upload_path location.
df = spark.readStream.format('cloudFiles') \
  .option('cloudFiles.format', 'csv') \
  .option('header', 'true') \
  .option('cloudFiles.schemaLocation', schema_path) \
  .schema('city string, year int, population long') \
  .load(upload_path)

# Start the stream.
# Use the checkpoint_path location to keep a record of all files that
# have already been uploaded to the upload_path location.
# For those that have been uploaded since the last check,
# write the newly-uploaded files' data to the write_path location.
df.writeStream.format('delta') \
  .option('checkpointLocation', checkpoint_path) \
  .option("mergeSchema","true") \
  .toTable('uc_streaming.lb_stream_demo.stream_pop')
```


14. Attempt to run the code interactively.
15. Create a Job with a UC enabled Job Cluster. Set the Security Mode to “Single-User” and assign a principal to the Job.

Scenario 2: Using an external table as a target for streaming/Autoloader

This is a similar scenario to the first scenario. In contrast to the first scenario, the target table will be an “External Table”. External Tables are created in a specified location on a cloud storage account. When upgrading an external table from legacy “Hive Metastore” to Unity Catalog, we don’t need to clone the data. The data stays intact and we just create a table in UC pointing to the location of the original table.

For this upgrade we will use the following example:

Cmd 3

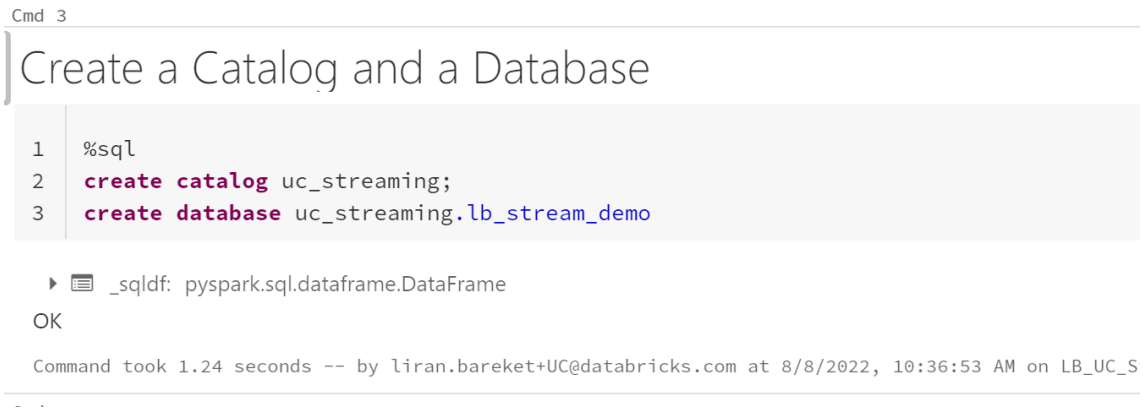
```
checkpoint_path = '/tmp/lb_streaming_ext/_checkpoints'
schema_path = '/tmp/lb_streaming_ext/_schema'
# checkpoint_path = 's3://databricks-ifi/sales-bucket/Autoloader/_checkpoints'
upload_path = 's3://databricks-ifi/sales-bucket/LB_AutoLoader_Input'

# Set up the stream to begin reading incoming files from the
# upload_path location.
df = spark.readStream.format('cloudFiles') \
  .option('cloudFiles.format', 'csv') \
  .option('header', 'true') \
  .option('cloudFiles.schemaLocation', schema_path) \
  .schema('city string, year int, population long') \
  .load(upload_path)

# Start the stream.
# Use the checkpoint_path location to keep a record of all files that
# have already been uploaded to the upload_path location.
# For those that have been uploaded since the last check,
# write the newly-uploaded files' data to the write_path location.
df.writeStream.format('delta') \
  .option('checkpointLocation', checkpoint_path) \
  .option("mergeSchema","true") \
  .start('s3://databricks-ifi/sales-bucket/tables/stream_pop_ext')
```

```
Cancel
▸ (1) Spark Jobs
▸ 92dbc14b-8f51-49df-a1ba-fa0e191ce345  Last updated: 15 seconds ago
▸ df: pyspark.sql.dataframe.DataFrame = [city: string, year: integer ... 1 more field]
Out[6]: <pyspark.sql.streaming.query.StreamingQuery at 0x7f3668cf5400>
```

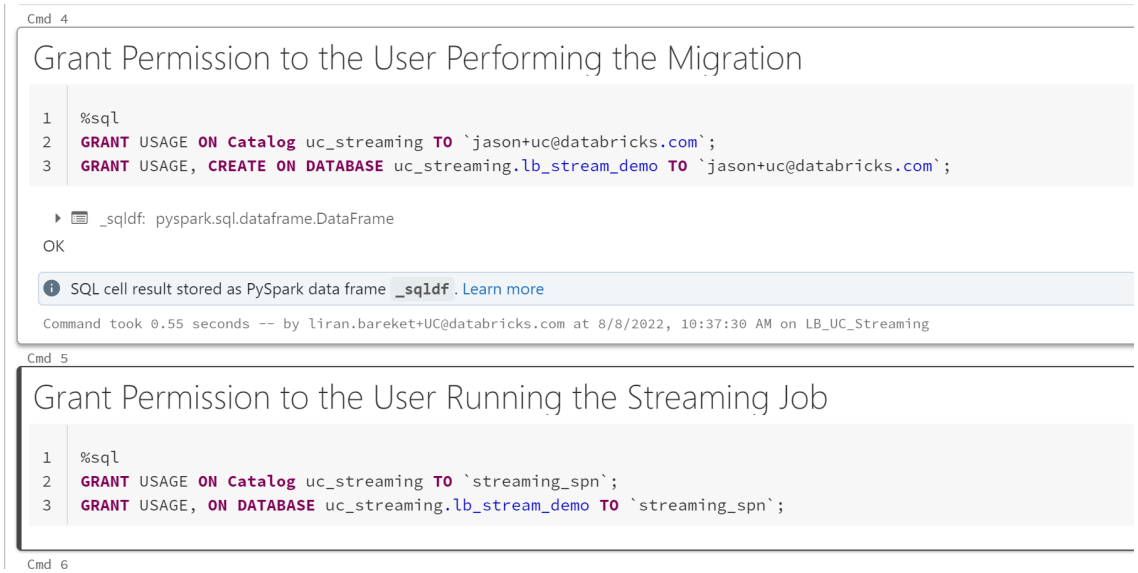
- This is an Autoloader example that reads csv files and saves to an S3 location directly (the "start" option). Another similar option will be where we specify toTable(<External Table>). To find out whether a table we use is an external table or managed, we use the `describe table extended <tablename>` SQL command
- We specify the checkpoint location as a dbfs location, we specify a schema location as well.

Upgrade Steps:

1. Create a Unity Catalog enabled cluster (DBR 11.1 or later, "Single-User" Security Mode).
2. The following steps will be performed by the **Metastore Admin**.
3. Create CREDENTIAL via the Data Explorer UI in DBSQL documentation
4. Create EXTERNAL LOCATION, either via the Data Explorer UI in DBSQL or with the below command documentation
5. Grant Permission to CREATE TABLES, READ FILES, and WRITE FILES to the user who will run the streaming job and the user performing the upgrade. It is recommended that a principal will run the job rather than a user.
6. Create a Catalog and Database for the target table (or use the ones you created during the "Designing Your Catalog Layout" step).

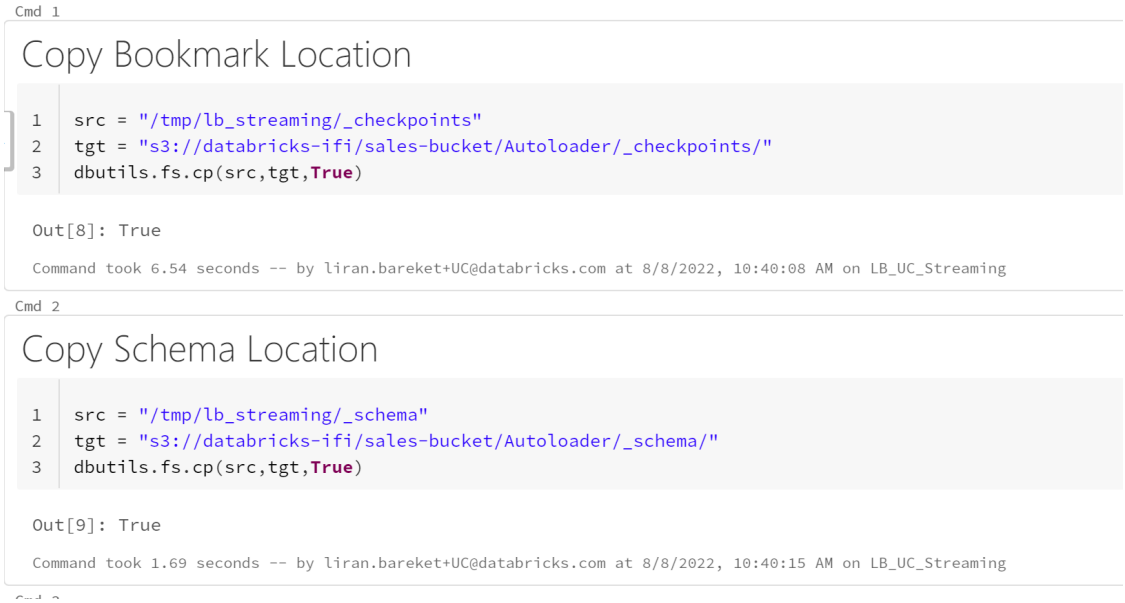


7. Grant access to the user performing the upgrade as well as to the user/principal running the streaming job.

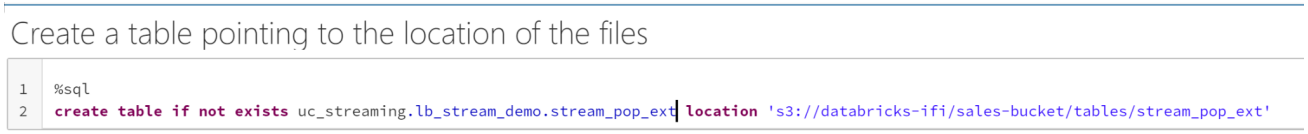


8. The following steps can be performed by a **developer**. The developer has to be granted CREATE TABLES, READ FILES, and WRITE FILES rights on the external location. The users have to be granted access to a UC Database or permission to create one.
9. Stop the stream. To assure a seamless transition to the new target, we will have to stop the stream.

10. Copy the checkpoint location to a new location on your cloud storage account. The new location has to be under an "External Location". See Documentation



11. Create an external table pointing to the same location of the original External Table/Stream Output Location.



12. Deprecate the old table. There are few options for deprecating the old tables. You can rename the table. Remove all the references to the old table.
13. Refactor the old streaming code. Apply the following change to the streaming code:
    a. Point the checkpoint path to the newly created path
    b. Point the schema path to the newly created path
    c. Point the toTable target to the newly cloned table. If the original stream used the "start" option, convert it to toTable, specifying the newly created table.

Cmd 7

```
checkpoint_path = 's3://databricks-ifi/sales-bucket/Autoloader/ext/_checkpoints/'
schema_path = 's3://databricks-ifi/sales-bucket/Autoloader/ext/_schema/'
# checkpoint_path = 's3://databricks-ifi/sales-bucket/Autoloader/_checkpoints'
upload_path = 's3://databricks-ifi/sales-bucket/LB_AutoLoader_Input'

# Set up the stream to begin reading incoming files from the
# upload_path location.
df = spark.readStream.format('cloudFiles') \
  .option('cloudFiles.format', 'csv') \
  .option('header', 'true') \
  .option('cloudFiles.schemaLocation', schema_path) \
  .schema('city string, year int, population long') \
  .load(upload_path)

# Start the stream.
# Use the checkpoint_path location to keep a record of all files that
# have already been uploaded to the upload_path location.
# For those that have been uploaded since the last check,
# write the newly-uploaded files' data to the write_path location.
df.writeStream.format('delta') \
  .option('checkpointLocation', checkpoint_path) \
  .option("mergeSchema","true") \
  .toTable('uc_streaming.lb_stream_demo.stream_pop_ext')
```

Cancel Running command

14. Attempt to run the code interactively.
15. Create a Job with a UC enabled Job Cluster. Set the Security Mode to "Single-User" and assign a principal to the Job.

Upgrade ML Workloads to Unity Catalog

Upgrade Scenarios:

Here are a few scenarios to look for in order to upgrade

1. Cluster running DBR less than 11.1
2. Cluster running DBR 11.1 and above

Limitations:

- DBR 11.1 cluster or later is required for UC.
- **Shared clusters are not supported** by ML Runtimes. This means that shared clusters cannot be used with auto ML, the feature store, and MLflow. **Single user clusters are supported**. In the near future service principal clusters will support ML Runtimes and will offer an option for sharing clusters.

Prerequisites

- UC should be enabled for the account
- At least one metastore has to be created and assigned to the workspace
- Catalogs/Databases/Tables have to be created and assigned proper ACLs

Upgrade Instructions:

Scenario 1: Clusters running DBR less than 11.1

With this scenario you will have to tear down the cluster and rebuild it, there's no way to upgrade clusters to use UC for this scenario. Follow instructions in scenario 2 to get clusters to UC. Of course, this means potentially re-testing workloads on DBR 11.1, if migrating from an older version of the runtime.

Scenario 2: Clusters running DBR 11.1+

Only one security mode will support ML runtimes.

- **Single user:** This mode allows you to Run SQL, Python, R and Scala workloads as a single user, with access to data secured in Unity Catalog. This mode supports the ML Runtime and all the common ML Libraries, and supports auto ML, the Feature Store, MLflow and so on. It should work as "Standard" clusters have in the past for ML workloads.

Upgrade DBSQL workloads to Unity Catalog

Upgrade Scenarios

We will look into a few scenarios that require migration.

1. Upgrading Managed and External Tables to UC
2. Upgrading ACLs to UC
3. Repointing Queries from Hive Metastore to UC

Upgrade Challenges

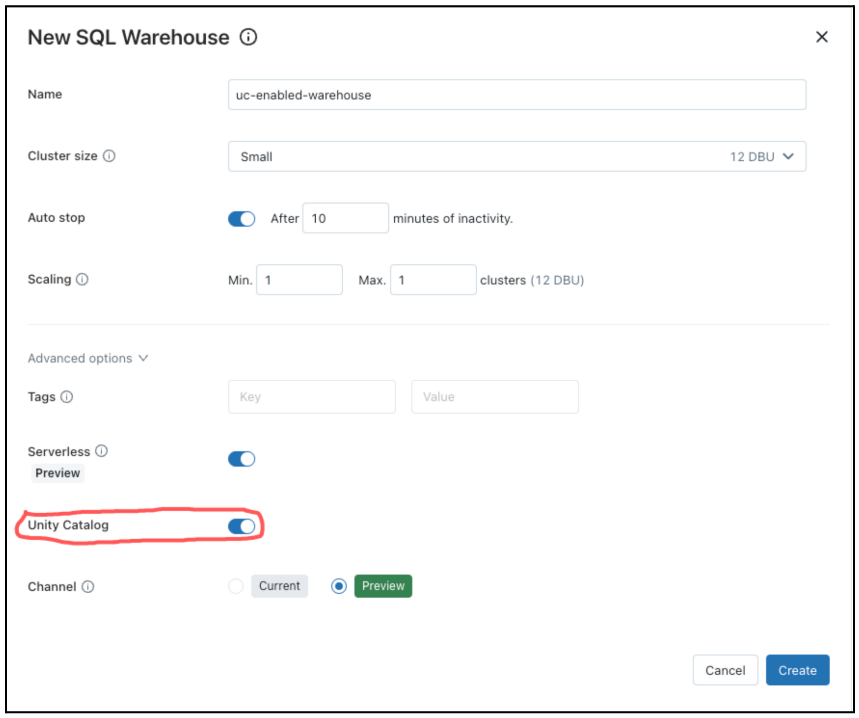
The biggest challenges are

1. A lot of the work has to be done manually and cannot be automated easily. Hence the team doing the upgrade needs to document their tasks thoroughly before and after migrating their workloads to UC.
2. While changing queries to point to the new UC table, it helps if the database names aren't changed. You can just add 'USE <catalog name>' at the top of the query and the rest of the query doesn't change. But if you change database names as well. You might have to touch every query to change database.table.

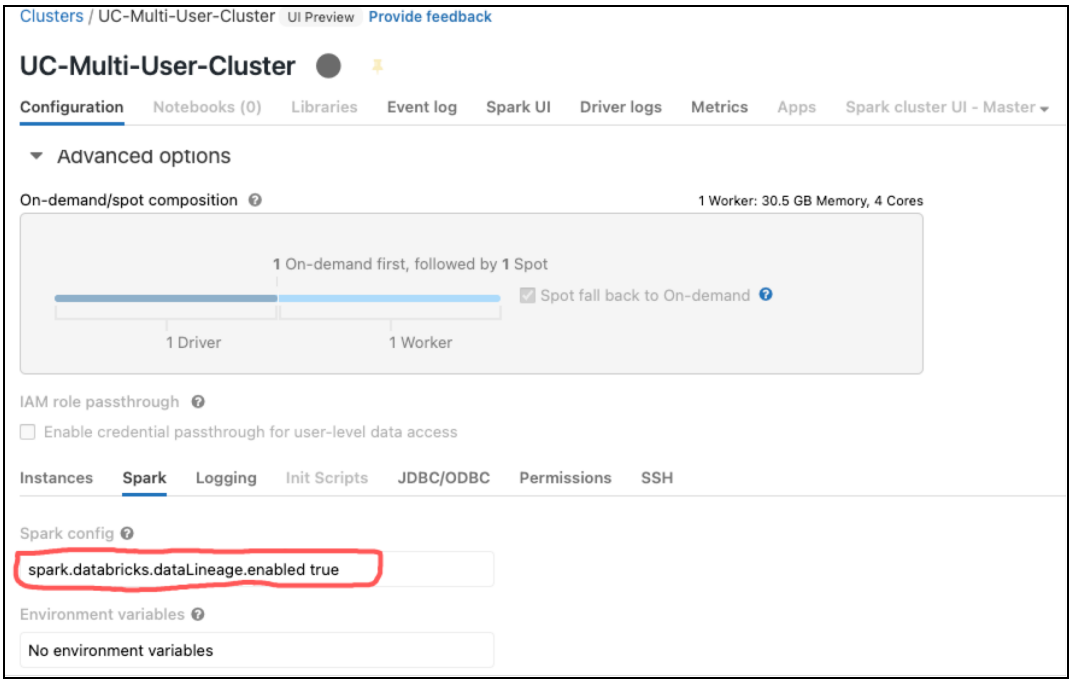
Pre-requisites

- UC should be enabled for the Account.
- Metastore has to be created and assigned to the Workspace by the Account Admin.

- Catalogs/Databases have to be created. To make the migration easier keep the database names same as those in hive_metastore
- An SQL Warehouse and Data Engineering Cluster has been created and spun up. While creating the SQL Warehouse ensure you enable Unity Catalog.



While creating a Data Engineering Cluster ensure you enable Lineage using a spark config setting under Advanced Options as shown below.



- A storage credential with an IAM role that authorizes Unity Catalog to access the external tables' location path. This needs to be created by a Databricks Account Admin.
- An external location that references the storage credential you just created and the path to the data on your cloud tenant where the external tables are stored. You need to be a metastore admin or a user with the CREATE EXTERNAL LOCATION privilege to execute this step.
- **CREATE TABLE** permission on the external location of the tables to be upgraded. (This is required because we will be creating tables in the Unity Catalog Metastore pointing to this location). This can be given to any user who needs to execute the upgrade.

1. Upgrading Managed and External Tables to UC

First make a list of all the tables that need to be upgraded from all the workspaces and databases that are under your management. You can automate this process using python scripts on a data engineering cluster such as shown here.

```
catalog= "hive_metastore"
database= "datawarehouse"
spark.sql(f"USE {catalog}.{database}")
table_list = spark.catalog.listTables(f"{database}")
```

Output:

[Table(name='planes', database='uc_abe_demo', description=None, tableType='MANAGED', isTemporary=False)]

This list will provide tableType using which you can figure out how many MANAGED and how many EXTERNAL tables you have.

If it's a MANAGED table you can use SQL to migrate it to Unity Catalog (UC) Metastore.
The SQL will look like below

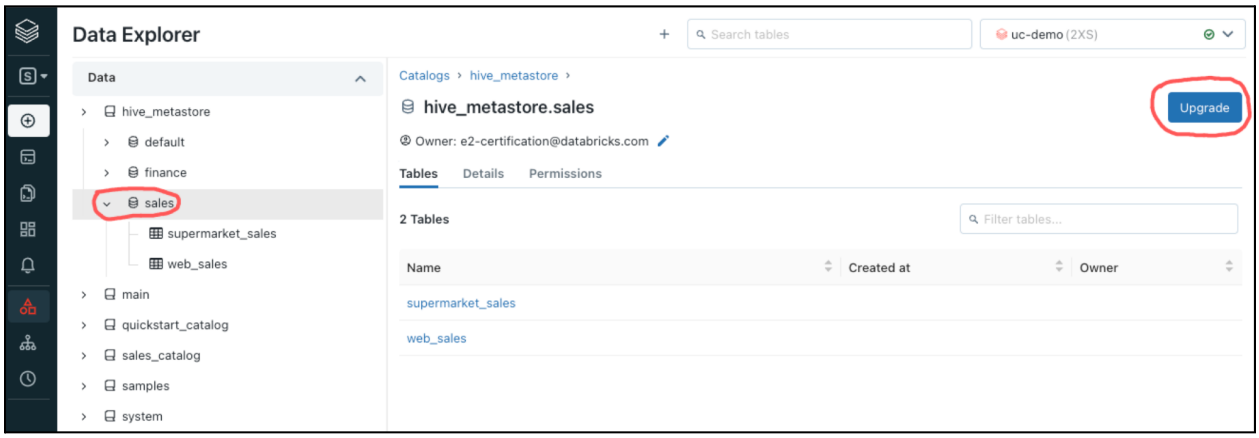
```
CREATE TABLE <catalog>.<new_schema>.<new_table>
AS SELECT * FROM hive_metastore.<old_schema>.<old_table>;
```

If it's an EXTERNAL table you can either do it using SQL or the DBSQL Data Explorer. Please note that this upgrade does not work with External tables with location specified as a DBFS location.

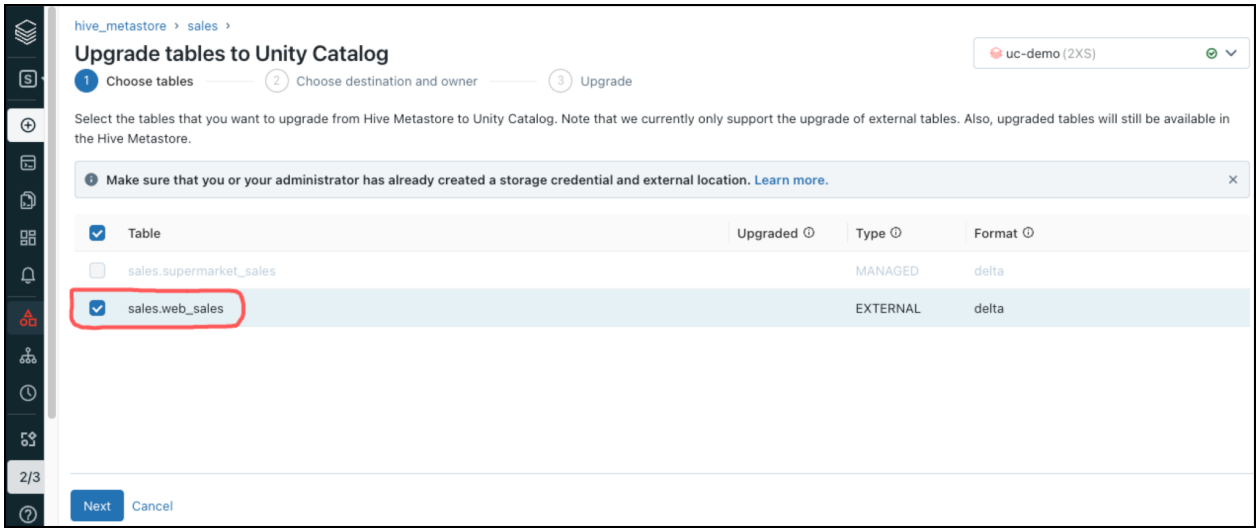
The SQL syntax example below to upgrade hive_metastore.sales.web_sales to sales_catalog.sales_db.web_sales.

```
CREATE TABLE sales_catalog.sales_db.web_sales LIKE hive_metastore.sales.web_sales COPY LOCATION;
```

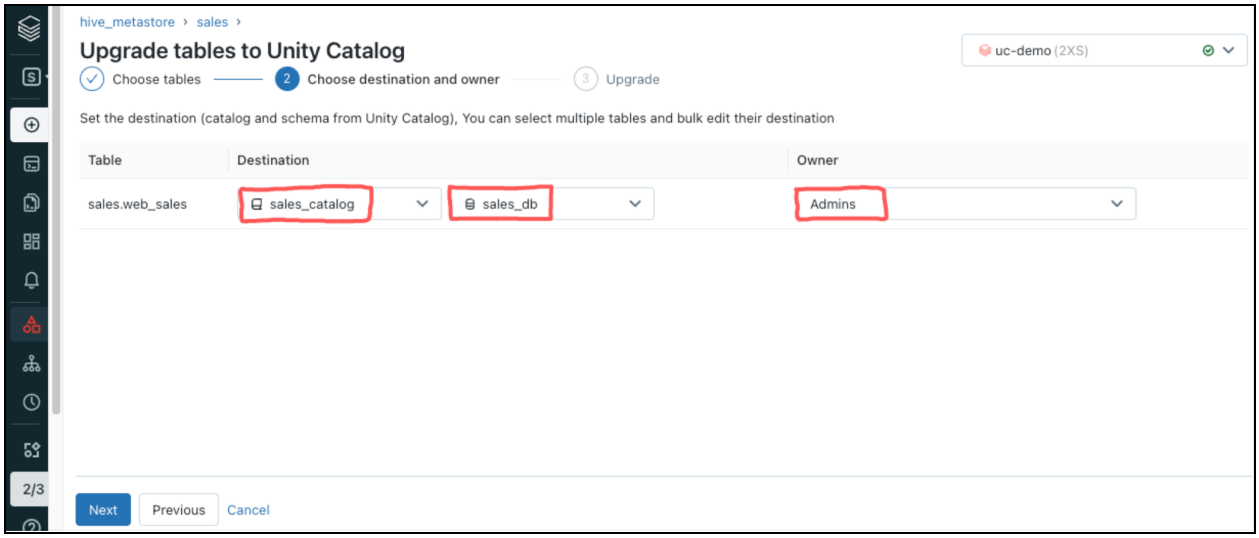
The DBSQL Data Explorer provides an "Upgrade" button for external tables as shown below.



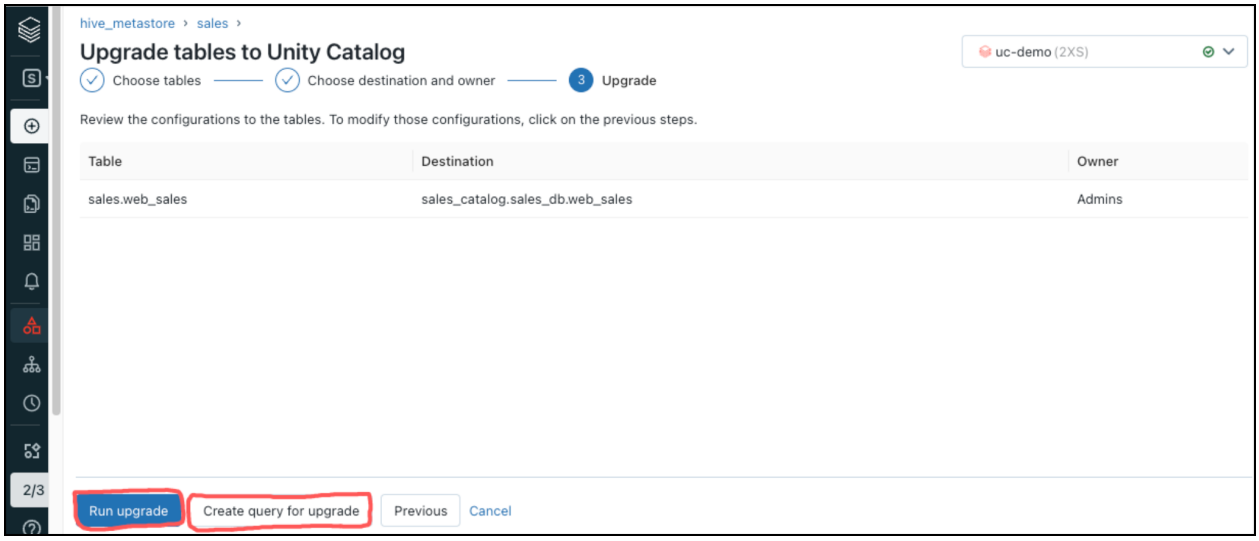
You can pick and choose the EXTERNAL tables within the Database/Schema to upgrade.



Next you choose the destination Catalog and Schema and the ownership group for the table



Finally you get to choose whether to upgrade it or just get the SQL Query to upgrade the table



This results in an EXTERNAL table being created in the Unity Catalog Metastore pointing to the EXTERNAL LOCATION.

After the tables have been created recreate the views that go on top of these tables.

2. Upgrading ACLs to UC

Once all the tables have been migrated to Unity Catalog Metastore you need to set the right access controls. First make a note of all the access privileges that have been set on the existing hive metastore tables. Please note that in Unity Catalog, access is NOT inherited by Tables based on access set at Database level e.g. this command won't work in UC enabled workspace if the intent is to provide select access to all tables within a schema to a user.

```
GRANT SELECT ON SCHEMA <schema-name> TO `<user>@<domain-name>`
```

All grants have to be provided explicitly at Table level. This will be changing when we release a new feature called "Bulk Grants" which will allow setting of table privileges at Database/Schema level.

```
SHOW GRANTS ON <securable_type> <securable_name>;
Eg: SHOW GRANTS ON hive_metastore.sales.web_sales
```

Start with providing USAGE access to the catalog and schema.

```
GRANT USAGE ON <catalog> TO <user/group>;
GRANT USAGE ON <schema> TO <user/group>;
```

Next setup access to the individual tables and views

```
GRANT <privilege> ON <securable_type> <securable_name> TO <principal>
```

3. Repointing Queries to UC enabled Tables

There are four ways to repoint your queries to the new UC Tables

1. Add USE <Catalog> above each query.
2. Change the table names to add catalog names in front of the schematable.. I.e. catalog.schema.table
3. Set the default catalog to a specific catalog at cluster level
4. Set the default catalog for the whole workspace

E.g. Let's say we upgrade supermarket_sales table from **hive_metastore.sales** database to **sales_catalog.sales** UC Schema/Database.

1. Just add USE CATALOG <catalog name> at the top of the query and the rest of the query remains unchanged. This is a recommended way in most cases as you don't need to change anything in the query itself. Also gives you the freedom to point your queries to different catalogs and test it out. (e.g. if you have different catalogs for dev, test and prod data)

```
USE CATALOG sales_catalog;

SELECT City,COUNT(*)
FROM sales.supermarket_sales
GROUP BY City;
```

2. Search and Replace all occurrences of table names to add <catalog name> in front of the table name. This option requires the most work so this should be the last resort if none of the other options are feasible.

```
SELECT City,COUNT(*)
FROM sales_catalog.sales.supermarket_sales
```

```
GROUP BY City
```

3. If you are using Data Engineering or Data Science clusters instead of the SQL Warehouse you can set the default catalog using the spark config as shown below. Obviously this works only in

```
spark.databricks.sql.initial.catalog.name <catalog-name>
```

4. If a majority of your queries in a workspace point to the same catalog then you can make that catalog the default catalog for the workspace using CLI commands as shown below.

```
databricks unity-catalog metastores assign —workspace-id 1234567890123456 \
                                          —metastore-id 7890-1cd2-3456-e789f0a12b34 \
                                          —default-catalog-name sales_catalog
```

Upgrade Fivetran Jobs to use Unity Catalog

Background

This document provides a solution for upgrading Fivetran Jobs to use Unity Catalog. Currently, the Fivetran connector for Databricks does not support 3-level namespacing. In order to overcome this, you must select a default catalog by specifying the cluster configuration on the interactive cluster that Fivetran is using. **This is not possible currently with DBSQL endpoints.**

This document assumes that you are **using a Databricks cluster** to run. The primary change in configuration is to update the Fivetran cluster.

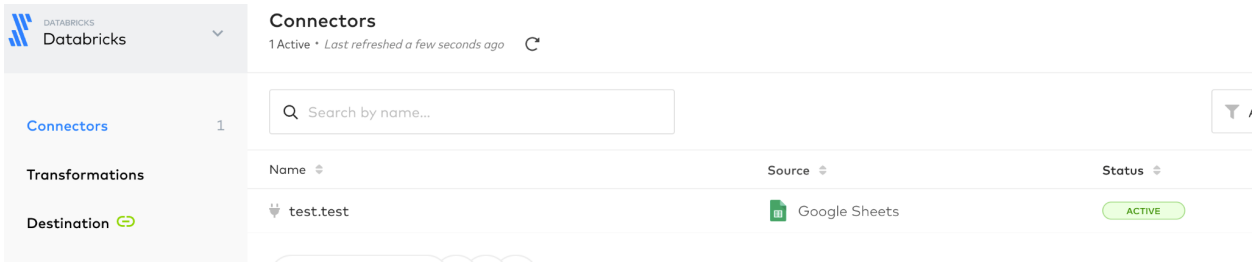
Upgrade Overview

The process involves these high-level steps.

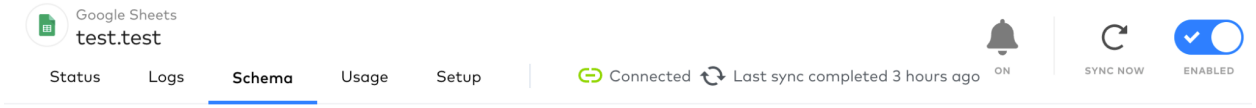
- For each Databricks "destination" in fivetran - turn off all existing Fivetran pipelines to Databricks
- Upgrade all fivetran tables in Hive Metastore to Unity Catalog
- Update the fivetran cluster
- Restart fivetran jobs

Step 1 - Turn off all existing Fivetran Jobs

Perform this step for each Databricks "destination" in Fivetran. A destination is a single cluster that your pipeline/s are writing to. We recommend that you do this all at once,



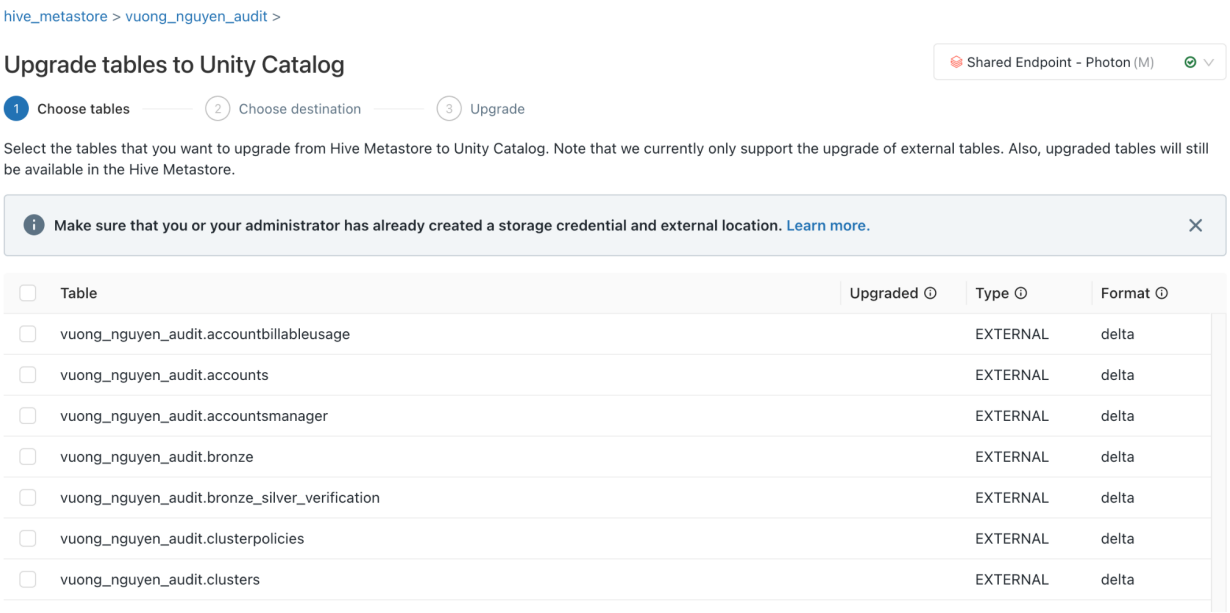
Click Enabled to disable the Job



Step 2 - Upgrade all Fivetran tables in Hive Metastore to Unity Catalog

Your Fivetran processes were previously writing data to a series of databases in Hive, they will now write to the same databases in Unity Catalog, under a **specific catalog of your choice.** Let's call this the "fivetran" catalog for now.

Use the Upgrade Wizard in the UI to upgrade your tables from Hive into your "fivetran" catalog. Note this will only work on **external** tables.



Step 3 - Update the fivetran cluster

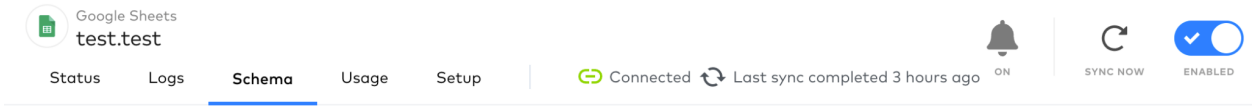
Edit the cluster config for your cluster and add the following spark configuration. Replace "fivetran" with the catalog name of your choice. This solution is not compatible with DBSQL clusters.

```
spark.databricks.sql.initial.catalog.name fivetran
```

Restart your cluster if it is running.

Step 4 - Restart fivetran jobs

Click enabled to restart the job.



Upgrade your DBT

DBT Cloud

Check out the announcement of the DBT Cloud integration with the Databricks Unity Catalog. Also, follow this link to upgrade your DBT Cloud using this migration guidance.

Step 7 - Setting up Delta Sharing (optional)

Delta Sharing

Accessing files across regions or clouds is facilitated by "Delta Sharing." First you have to enable Delta Sharing for your Databricks account; the instructions are shown here.

Databricks Delta Sharing Outside of Databricks

This is also referred to as "Open Sharing" in the Databricks documentation. The instructions can be found here to set up a recipient. Also here are instructions on how to read data from an Open Delta Share. Note that if the Delta Share comes from another Databricks account, see instructions below instead.

Databricks to Databricks (D2D) Delta Sharing

The instructions for enabling D2D Delta Sharing can be found here. Additionally, these instructions explain how to read data from a D2D Delta Share.

Extra resources

Setting up UC workspaces with Terraform

You can automate the Unity Catalog enabled infrastructure setup or add to your UC enabled account by using the Databricks Terraform provider.

AWS

Note the pre-requisites in this Databricks UC Terraform guide.

Additionally, here is a Terraform script for deploying databricks on AWS with UC.

Azure

Note the pre-requisites in this Databricks UC Terraform guide.

GCP

None as of yet.