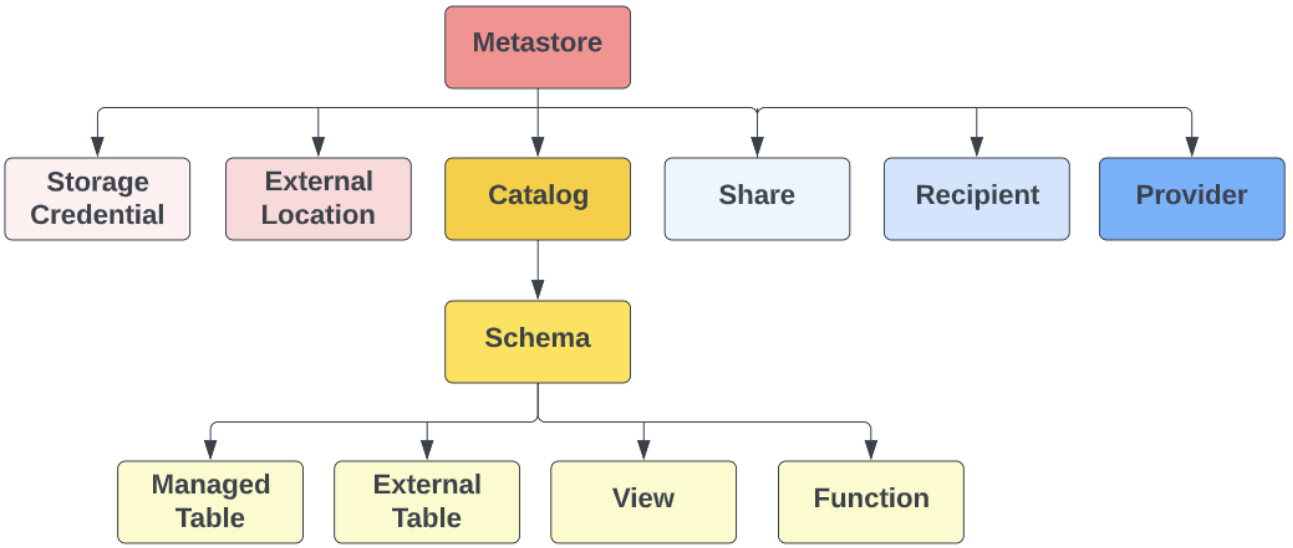
Chefs

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

Ingredients & Tools



Terms	Definition
Unity Catalog	Unity Catalog (UC) is a unified governance solution for all data and AI assets including files, tables, machine learning models, and dashboards in your lakehouse.
Metastore	A metastore is the top-level container of objects in the Unity Catalog. It stores data assets (tables and views) and the permissions that govern access.
Catalog	The first layer of the object hierarchy which is used to organize your data assets.
Schema	Schemas (aka databases) are the second layer of the object hierarchy and contain tables and views.
Table	The lowest level in the object hierarchy, tables can be external (stored in external locations in your cloud storage of choice) or managed (stored in a storage container in your cloud storage that you create expressly for Databricks)
View	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.
Materialized View	An automatically refreshed view that helps to improve query latency by pre-computing queries and/or frequently used computation. (In Private Preview)

Function	A user-defined function that is contained within a schema.
External Location	An object that combines a cloud storage path with a storage credential in order to authorize access to the cloud storage path.
Storage Credential	Encapsulates a long-term cloud credential that provides access to cloud storage.
Provider	An entity that has made data available for sharing.
Share	A logical grouping for the tables you intend to share.
Recipient	Identifies an organization with which you want to share any number of shares.

Division of Labor

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

	Can do
	Should do

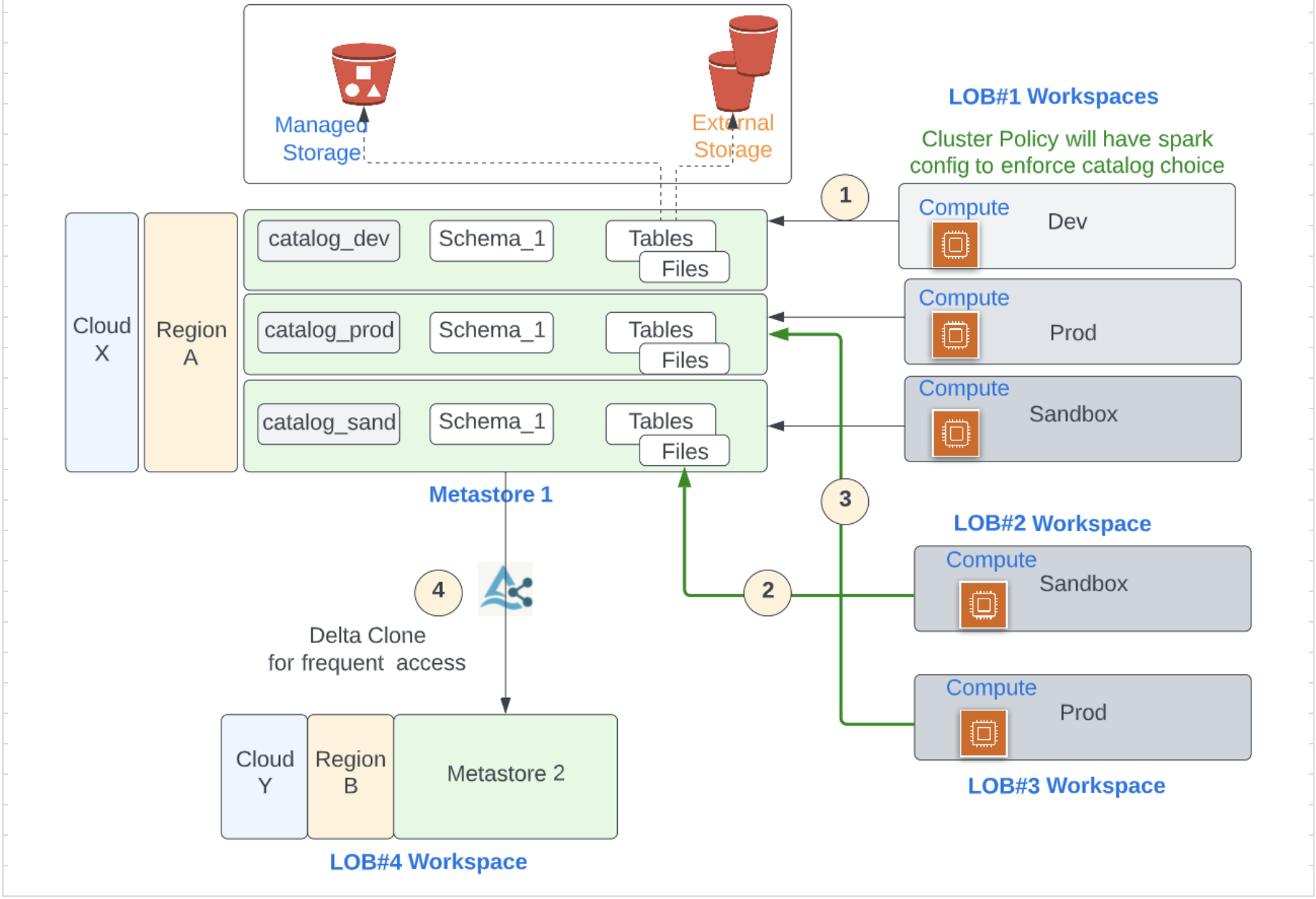
Task	Status
Create Metastore	☐
Manage Principals	☐
Setup SSO	☐
Create Catalog	☐
Create Storage Credential	☐
Create External Location	☐
Delegate Access to Data	☐
Manage Compute Access	☐
Assign Principals to Workspace	☐

Cooking Steps

Task	Persona	Detail	Status
Collaborate with Identity Admin; Identify Admin Personas			
Set up SCIM from IDP	**Account Admin (+ Identity Admin)**	AWS, Azure	☐
Set up SSO		AWS, Azure	☐
Identify Core Admin Personas (Account, Metastore, Workspace)		AWS, Azure	☐
Identify Recommended Admin Personas (Catalog, Compute, Schema)		See Introducing the Chefs section above	☐
Collaborate with Cloud Admin; Create Cloud Resources			
Create Root bucket	**Account Admin (+ Cloud Admin)**	AWS, Azure	☐
Create IAM role (AWS) Create Access Connector Id (Azure)		AWS, Azure	☐
Create a Metastore			
Create Metastore	**Account Admin**	AWS, Azure	☐
Transfer ownership of metastore to the metastore_admin group	**Account Admin**	AWS, Azure	☐
Create Storage Credentials & External Locations (for external tables)			
Create Storage Credentials	**Account Admin**	AWS, Azure	☐
Create External Locations	**Metastore Admin**	AWS, Azure	☐
Make the workspace UC-enabled			
Create Workspace (if not exists)	**Account Admin (AWS) Cloud Admin (Azure)**	AWS, Azure	☐
Metastore Assignment (assign metastore to the WS)	**Account Admin**	AWS, Azure	☐
Workspace Permission Assignment (assign principals to WS)	**Workspace Admin**	AWS, Azure	☐
Create Catalog			

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

Scenario Examples



Scenario 1: LOB#1

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

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

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

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

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

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

Scenario 2: LOB#2

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

Scenario 3: LOB#3

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

Scenario 4: LOB#4

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