1) untiy catalog vs legacy hive metastore?

1) Unity Catalog supports Delta format tables (structured) via PySpark write.

It supports external tables with all formats (Delta, Parquet, CSV, JSON, etc.) using SQL only.

It also supports Volumes to upload and access files with governance.

2) Legacy Hive Metastore supports tables with all file formats (Delta, Parquet, CSV, JSON, etc.) using both SQL and PySpark write.

It does not support Volumes or advanced governance features.

2) in compute/cluster different types of accessmodes and it's use and limitations?

1) Standard (formerly: Shared)

It allows multiple users with strong isolation.

It is best for Unity Catalog-based workloads.

Does not support DBFS mounts.

Supports Unity Catalog and secure data access.

No credential passthrough or mounting.

2) Dedicated (formerly: Single User)

Runs as a single user (full isolation).

Supports all languages and ML workloads.

Supports DBFS mounting.

Good for development, testing, and access to legacy storage paths.

Shared across workspace users if reused.

3) No Isolation Shared (Legacy – not recommended)

Supports all languages.

Supports DBFS mounting.

No user isolation — if one user/session errors, it may impact others.

Useful for quick tests or batch jobs, not for production.

4) Credential Passthrough (Shared/Single User)

Allows Azure AD identity passthrough to access data securely.

Supports mounting and secure access.

Shared version → supports Python & SQL only.

Single user version → supports all languages.

5) Custom

Based on admin or user-defined configuration.

Can be used to meet specific security, compliance, or tuning needs.

Needs advanced knowledge; not typically used for general workloads.

3) how to mount and unmount, what are the steps for mounting?

Create entra -> app registration ->New registration by click +-> name->databricks -> choose default for now(you can change it accordingly) ->register/save. A screenshot of a computer

AI-generated content may be incorrect.

Goto resource group -> storage -> IAM -> add role assignement + -> Role -> job function role -> add storage blob contributor -> next ->members->managed identity->select memebers->enter etnra service name which created -> review and assign.

Goto entra -> select which created by you above -> manage -> certficates and secrets -> new client secret->add description select days to expire -> create ->after creating it will show value and secret id copy value and keep it secret id not required

And there it’s self goto same entra -> overview -> copy Application (client) ID and Directory (tenant) ID for below steps.

Gotog azure key vault -> create -> selct resourc group of databricks-> keyvault name -> region where databricks exists same-> select pricing tier -> select expire days -> next -> access configuration -> permission model -> vault access policy -> access policie -> create custom or take defuault one -> review+create.

After creating azure key vault -> goinside -> settings-> properties-> copy vault uri and resource ID.

Goto databricks

<https://adb-2344682967501175.15.azuredatabricks.net/#secrets/createScope>

here give name-> azure key vault -> DNS NAME paste here vault uri (which you copied previously key vault) -> resource id paste in resource id field->create

goto -> notebook-> dbutils.secrets.list(scope="bojjascope")(this one which we created scope)it will show all secrets which you have added in key vault and use that key to use it like below.

dbutils.secrets.get(scope="bojjascope",key="bojjastoragekey”)

1st approach to mount setup

configs = {"fs.azure.account.auth.type": "OAuth",

          "fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",

          "fs.azure.account.oauth2.client.id": "<Application (client) ID>",

          "fs.azure.account.oauth2.client.secret": dbutils.secrets.get(scope="bojjascope",key="bojjastoragekey"),(this one which we created scope)

          "fs.azure.account.oauth2.client.endpoint": "https://login.microsoftonline.com/<Directory (tenant) ID>/oauth2/token"}

# Optionally, you can add <directory-name> to the source URI of your mount point.

dbutils.fs.mount(

  source = "abfss://bojjacontainersrc@bojjastorage.dfs.core.windows.net/",

  mount\_point = "/mnt/bojjacontainersrc",

  extra\_configs = configs)

2nd to create storage and external location follow 9th step

3rd one setup

spark.conf.set("fs.azure.account.auth.type.bojjastorage.dfs.core.windows.net", "OAuth")

spark.conf.set("fs.azure.account.oauth.provider.type.bojjastorage.dfs.core.windows.net", "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider")

spark.conf.set("fs.azure.account.oauth2.client.id.bojjastorage.dfs.core.windows.net", "<Application (client) ID>")

spark.conf.set("fs.azure.account.oauth2.client.secret.bojjastorage.dfs.core.windows.net", "<secrets\_value>") or scope dbutils.secrets.get(scope="bojjascope",key="bojjastoragekey"),(this one which we created scope)

spark.conf.set("fs.azure.account.oauth2.client.endpoint.bojjastorage.dfs.core.windows.net", "https://login.microsoftonline.com/<Directory (tenant) ID>/oauth2/token")

dbutils.fs.ls('abfss://bojjacontainersrc@bojjastorage.dfs.core.windows.net/')

this will be only till session available.

Here’s a clean, step-by-step final version for

**3) How to Mount and Unmount Storage in Databricks? Steps for Mounting**

**Step 1: Register an App in Azure Entra ID (Azure AD)**

1. Go to **Azure Entra** (Azure AD) → **App registrations** → Click **+ New registration**.
2. Give it a **name** (e.g., databricks), choose default settings or below options, then click **Register**.
3. A screenshot of a computer

   AI-generated content may be incorrect.
4. In the newly created app, note down:
   * **Application (client) ID**
   * **Directory (tenant) ID**

**Step 2: Assign Azure Storage Blob Contributor Role**

1. Go to **Azure Portal** → **Resource Group** → Select your **Storage Account**.
2. Navigate to **Access control (IAM)** → **Add role assignment**.
3. Assign the role **Storage Blob Data Contributor** to the **Managed Identity** of the Azure AD app created above.

**Step 3: Create Client Secret in Azure Entra ID**

1. In your app registration, go to **Certificates & secrets**.
2. Click **New client secret**, add description and expiration, then click **Add**.
3. Copy the **Value** of the secret (not the secret ID) — keep it safe as it won't be shown again.

**Step 4: Set Up Azure Key Vault**

1. Create an **Azure Key Vault** in the same region as Databricks.
2. Set Access Policies to allow your app or user to **Get** secrets.
3. Note the **Vault URI** and **Resource ID**.
4. In Databricks, create a secret scope that links to Azure Key Vault using UI or CLI:
5. https://<databricks-instance>#secrets/createScope
   * Scope name: e.g., bojjascope
   * DNS name: Vault URI
   * Resource ID: Key Vault Resource ID
6. Verify secrets inside Databricks notebook:
7. dbutils.secrets.list(scope="bojjascope")
8. dbutils.secrets.get(scope="bojjascope", key="bojjastoragekey")

**Step 5: Mount the Storage in Databricks Notebook**

Use OAuth configs with the secret stored in Key Vault:

configs = {

"fs.azure.account.auth.type": "OAuth",

"fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",

"fs.azure.account.oauth2.client.id": "<Application (client) ID>",

"fs.azure.account.oauth2.client.secret": dbutils.secrets.get(scope="bojjascope", key="bojjastoragekey"),

"fs.azure.account.oauth2.client.endpoint": "https://login.microsoftonline.com/<Directory (tenant) ID>/oauth2/token"

}

dbutils.fs.mount(

source = "abfss://bojjacontainersrc@bojjastorage.dfs.core.windows.net/",

mount\_point = "/mnt/bojjacontainersrc",

extra\_configs = configs

)

**Step 6: Verify Mount**

dbutils.fs.ls("/mnt/bojjacontainersrc")

**Step 7: Unmount (if needed)**

dbutils.fs.unmount("/mnt/bojjacontainersrc")

**Additional Approach: Using Spark Configs (Temporary for Session)**

spark.conf.set("fs.azure.account.auth.type.bojjastorage.dfs.core.windows.net", "OAuth")

spark.conf.set("fs.azure.account.oauth.provider.type.bojjastorage.dfs.core.windows.net", "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider")

spark.conf.set("fs.azure.account.oauth2.client.id.bojjastorage.dfs.core.windows.net", "<Application (client) ID>")

spark.conf.set("fs.azure.account.oauth2.client.secret.bojjastorage.dfs.core.windows.net", dbutils.secrets.get(scope="bojjascope", key="bojjastoragekey"))

spark.conf.set("fs.azure.account.oauth2.client.endpoint.bojjastorage.dfs.core.windows.net", "https://login.microsoftonline.com/<Directory (tenant) ID>/oauth2/token")

# Access the storage directly (valid only for the session)

dbutils.fs.ls('abfss://bojjacontainersrc@bojjastorage.dfs.core.windows.net/')

4) data lake vs delta lake?

Data Lake

✅Supports all file formats: structured, semi-structured, and unstructured (e.g., CSV, JSON, Parquet, images, etc.).

💾 Acts as a centralized storage layer (usually using services like Azure Data Lake Storage or AWS S3).

❌ No built-in ACID transactions, versioning, or schema enforcement.

⚙️ Mostly used for raw data ingestion and storage, not for advanced analytics or transactions.

🔹 Delta Lake

✅ An open-source storage layer built on top of data lakes.

✅ Supports ACID transactions, schema enforcement, time travel, data versioning, and concurrent writes.

✅ Enables reliable streaming and batch data processing.

🧠 Acts more like a data warehouse but runs on data lake storage.

🔸 Key Point

Databricks is built on top of a data lake, and adds Delta Lake capabilities

to provide data warehouse–like features on open data lake storage.

5) how to create secrets scope?

Follow 3rd step already covered in that 3rd step.

6) managed table and external table?

🔹 Managed Table

✅ Data + metadata are fully managed by Databricks.

❌ No external location is specified during creation.

📂 Data is stored in default managed storage.

🗑️ When you drop the table, both data and metadata are deleted.

🧠 Suitable for temporary or internal datasets.

🔹 External Table

✅ You specify an external location (e.g., ADLS, S3) to store the table data.

📂 Only metadata is managed by Databricks; data files stay in the external storage.

🗑️ When you drop the table, only metadata is deleted, data remains intact.

✅ Useful for long-term storage, external data sharing, or cross-tool access.

🔸 Unity Catalog Specifics

✅ In Unity Catalog, tables are managed by default.

🔐 To create external tables, you must:

Create a Storage Credential.

Register an External Location.

Use that location when creating the external table.

7) which clusters need to use for job runs?

🔹 Job Clusters

✅ For scheduled or triggered jobs, Job Clusters should be used.

🎯 Created directly from the Jobs UI while configuring a job.

📦 Ephemeral clusters – they are created at job start and terminated after completion, saving cost.

🧼 Automatically isolated per job run — ensures clean environment each time.

🔒 Ensures better resource isolation, security, and cost control compared to interactive clusters.

🔸 Why Not Use All-Purpose Clusters for Jobs?

❌ All-purpose clusters stay running and are shared — can incur higher cost and lack isolation.

✅ Only suitable for ad-hoc notebooks, development, or collaborative work.

8) how to create external table in unity catalog what are steps involved with azure?

**9) How to Create Storage & External Location for Workspace in Databricks?**

**🔹 Step 1: Create a Storage Credential**

1. Go to **Catalog Explorer**.
2. Click on **“+” icon** or the **gear/settings icon** ➝ select **"Storage Credential"**.
3. Enter a **name** for the storage credential.
4. In **Azure**:
   * Use an existing **Access Connector for Azure Databricks**.
   * Go to the Azure portal ➝ open the **Access Connector** ➝ copy the **Resource ID**.
   * Paste it into the **Access Connector field** in Databricks.
5. Click **Save**.

**🔹 Step 2: Create an External Location**

1. In **Catalog Explorer**, again click **“+”** ➝ select **"External Location"**.
2. Enter a **name**.
3. In the **Storage path** field, enter the full path:
4. abfss://<container>@<storage\_account>.dfs.core.windows.net/<optional-path>
5. Select the **Storage Credential** created in step 1.
6. Click **Save**.

**🔹 Step 3: Assign Permissions**

1. Go to the created **Storage Credential** and **External Location** entries in Catalog Explorer.
2. Click **"Permissions"** ➝ **"Grant"**.
3. Choose **user**, **admin**, or **group name**.
4. Select **“ALL PRIVILEGES”** or specific permissions (e.g., READ FILES, WRITE FILES).
5. Click **Save**.

**steps to create or follow to setup data bricks.**

1) how to setup/create databricks in azure?

2) how to give storage container access to mounting in databricks?

3) what is the use of access connector and how to setup?

4) how to share unity catalog table in databricks and steps?

5) what is the use of entra id and key vaault, how to use it?

6) how to create secrets in azure and steps?

7) access connector vs entra id?

8) what is IAM and how you are managing to give permissions and for which are service we need to give, while to setup databricks?

9) using pyspark dataframe can we create external table in untiy catalog?

NO.