**Step 3: Unity Catalog + storage credentials + first ingestion (DLT)** for your Databricks-first AML platform.

Below are **drop-in Terraform blocks** plus a **starter DLT notebook** to land your first datasets into **raw → silver → gold** under Unity Catalog.

**3A) Azure Access Connector (for UC) + permissions**

*Add to your existing Terraform (/infra) and apply in the same subscription/region.*

# ---------- Access Connector for Unity Catalog ----------

resource "azurerm\_databricks\_access\_connector" "uc" {

name = "dac-aml-uc"

resource\_group\_name = azurerm\_resource\_group.dbx.name

location = local.location

identity { type = "SystemAssigned" }

tags = local.tags

}

# Let UC's Managed Identity access ADLS (containers raw/silver/gold)

resource "azurerm\_role\_assignment" "uc\_sa\_contrib" {

scope = azurerm\_storage\_account.adls.id

role\_definition\_name = "Storage Blob Data Contributor"

principal\_id = azurerm\_databricks\_access\_connector.uc.identity[0].principal\_id

}

Apply:

terraform apply -auto-approve \

-var="tenant\_id=$(az account show --query tenantId -o tsv)" \

-var="sa\_name=<same-storage-name>"

**3B) Unity Catalog via Databricks provider**

Create a **new folder** /uc for UC resources.

**providers.tf**

terraform {

required\_providers {

databricks = { source = "databricks/databricks", version = "~> 1.45" }

}

}

# Account-level provider (for metastore)

provider "databricks" {

alias = "accounts"

host = "https://accounts.azuredatabricks.net"

account\_id = var.databricks\_account\_id

auth\_type = "azure-cli" # uses your az login

}

# Workspace-level provider (for catalogs/schemas/grants/pipeline)

provider "databricks" {

alias = "ws"

host = azurerm\_databricks\_workspace.ws.workspace\_url

azure\_workspace\_resource\_id = azurerm\_databricks\_workspace.ws.id

auth\_type = "azure-cli"

}

**variables.tf**

variable "databricks\_account\_id" { type = string } # find in Databricks Admin Console

variable "sa\_name" { type = string } # same ADLS from Step 1

**unity\_catalog.tf**

# ----- Create a UC Metastore and assign it to this workspace -----

resource "databricks\_metastore" "aml" {

provider = databricks.accounts

name = "ms-aml"

region = azurerm\_resource\_group.dbx.location # must match workspace region

storage\_root = "abfss://logs@${var.sa\_name}.dfs.core.windows.net/uc/ms-aml"

}

resource "databricks\_metastore\_assignment" "ws\_assign" {

provider = databricks.accounts

metastore\_id = databricks\_metastore.aml.id

workspace\_id = azurerm\_databricks\_workspace.ws.workspace\_id

default\_catalog\_name = "hive\_metastore" # or leave default; we’ll use catalog "aml"

}

# ----- Storage credential using Azure Access Connector MI -----

resource "databricks\_storage\_credential" "aml" {

provider = databricks.ws

name = "sc-aml"

azure\_managed\_identity {

access\_connector\_id = azurerm\_databricks\_access\_connector.uc.id

}

comment = "UC credential to ADLS via Access Connector"

}

# ----- External locations (one per container) -----

resource "databricks\_external\_location" "raw" {

provider = databricks.ws

name = "loc-raw"

url = "abfss://raw@${var.sa\_name}.dfs.core.windows.net/"

credential\_name = databricks\_storage\_credential.aml.name

}

resource "databricks\_external\_location" "silver" {

provider = databricks.ws

name = "loc-silver"

url = "abfss://silver@${var.sa\_name}.dfs.core.windows.net/"

credential\_name = databricks\_storage\_credential.aml.name

}

resource "databricks\_external\_location" "gold" {

provider = databricks.ws

name = "loc-gold"

url = "abfss://gold@${var.sa\_name}.dfs.core.windows.net/"

credential\_name = databricks\_storage\_credential.aml.name

}

# ----- Catalog & schemas mapped to the locations -----

resource "databricks\_catalog" "aml" {

provider = databricks.ws

name = "aml"

comment = "AML analytics catalog"

}

resource "databricks\_schema" "raw" {

provider = databricks.ws

catalog\_name = databricks\_catalog.aml.name

name = "raw"

comment = "Bronze landing"

storage\_location = databricks\_external\_location.raw.url

}

resource "databricks\_schema" "silver" {

provider = databricks.ws

catalog\_name = databricks\_catalog.aml.name

name = "silver"

comment = "Conformed layer"

storage\_location = databricks\_external\_location.silver.url

}

resource "databricks\_schema" "gold" {

provider = databricks.ws

catalog\_name = databricks\_catalog.aml.name

name = "gold"

comment = "Serving layer"

storage\_location = databricks\_external\_location.gold.url

}

# ----- Grants (adjust to your groups) -----

# Expect that your Entra groups are SCIM-synced into Databricks with same names:

# grp-aml-data-engineers, grp-aml-data-scientists, grp-aml-investigators

data "databricks\_group" "eng" { provider = databricks.ws, display\_name = "grp-aml-data-engineers" }

data "databricks\_group" "sci" { provider = databricks.ws, display\_name = "grp-aml-data-scientists" }

data "databricks\_group" "inv" { provider = databricks.ws, display\_name = "grp-aml-investigators" }

resource "databricks\_grants" "catalog" {

provider = databricks.ws

catalog = databricks\_catalog.aml.name

grant {

principal = data.databricks\_group.eng.display\_name

privileges = ["USE\_CATALOG", "CREATE\_SCHEMA"]

}

grant {

principal = data.databricks\_group.sci.display\_name

privileges = ["USE\_CATALOG"]

}

grant {

principal = data.databricks\_group.inv.display\_name

privileges = ["USE\_CATALOG"]

}

}

resource "databricks\_grants" "schema\_raw" {

provider = databricks.ws

schema = "${databricks\_catalog.aml.name}.${databricks\_schema.raw.name}"

grant { principal = data.databricks\_group.eng.display\_name privileges = ["USE\_SCHEMA","CREATE\_TABLE","MODIFY","SELECT"] }

}

resource "databricks\_grants" "schema\_silver" {

provider = databricks.ws

schema = "${databricks\_catalog.aml.name}.${databricks\_schema.silver.name}"

grant { principal = data.databricks\_group.eng.display\_name privileges = ["USE\_SCHEMA","CREATE\_TABLE","MODIFY","SELECT"] }

grant { principal = data.databricks\_group.sci.display\_name privileges = ["USE\_SCHEMA","SELECT","CREATE\_TABLE"] }

}

resource "databricks\_grants" "schema\_gold" {

provider = databricks.ws

schema = "${databricks\_catalog.aml.name}.${databricks\_schema.gold.name}"

grant { principal = data.databricks\_group.inv.display\_name privileges = ["USE\_SCHEMA","SELECT"] }

grant { principal = data.databricks\_group.sci.display\_name privileges = ["USE\_SCHEMA","SELECT"] }

}

Apply:

cd uc

terraform init

terraform apply -auto-approve \

-var="databricks\_account\_id=<your-dbx-account-id>" \

-var="sa\_name=<same-storage-name>"

*Result:* UC metastore bound to your workspace, storage credential via Access Connector, external locations for **raw/silver/gold**, catalog aml with schemas mapped and permissions set.

**3C) First ingestion with Delta Live Tables (DLT)**

**1) Create a Databricks notebook (e.g., /Repos/aml/dlt/transactions.py) with this starter:**

import dlt

from pyspark.sql.functions import col, to\_timestamp, regexp\_replace

# ---- Paths (UC external locations) ----

RAW = "abfss://raw@<STORAGE>.dfs.core.windows.net/transactions/"

SILVER= "abfss://silver@<STORAGE>.dfs.core.windows.net/transactions/"

GOLD = "abfss://gold@<STORAGE>.dfs.core.windows.net/features/"

@dlt.table(

name="aml\_raw\_transactions",

comment="Landing of transactions (auto loader).",

table\_properties={"quality": "bronze"}

)

def raw\_transactions():

return (

spark.readStream.format("cloudFiles")

.option("cloudFiles.format","json") # or "csv"/"parquet"

.option("cloudFiles.inferColumnTypes","true")

.load(RAW)

.withColumn("ingest\_ts", to\_timestamp(col("\_metadata.file\_modification\_time")/1000))

)

@dlt.view(name="txn\_clean")

def txn\_clean():

df = dlt.read\_stream("aml\_raw\_transactions")

return (

df

.withColumn("amount", col("amount").cast("decimal(18,2)"))

.withColumn("value\_date", to\_timestamp("value\_date"))

.withColumn("currency", regexp\_replace(col("currency"), r"[^A-Z]", ""))

)

@dlt.table(

name="aml\_transactions",

comment="Conformed transactions (deduped).",

table\_properties={"quality": "silver"}

)

def transactions\_silver():

return (

dlt.read\_stream("txn\_clean")

.dropDuplicates(["tx\_id"])

)

@dlt.table(

name="aml\_txn\_features",

comment="Simple AML features per account per day.",

table\_properties={"quality": "gold"}

)

def txn\_features():

from pyspark.sql.functions import sum as \_sum, countDistinct, date\_trunc

df = dlt.read("aml\_transactions")

return (

df.groupBy(

col("account\_id"),

date\_trunc("DAY", col("value\_date")).alias("d")

).agg(

\_sum("amount").alias("sum\_amt"),

countDistinct("counterparty\_id").alias("cp\_count")

)

)

Drop sample JSON/CSV files into raw/transactions/ (via Storage Explorer or ADF Copy) and DLT will incrementally ingest.

**2) Provision the DLT pipeline via Terraform (workspace provider)**

resource "databricks\_pipeline" "aml\_core" {

provider = databricks.ws

name = "dlt-aml-core"

catalog = databricks\_catalog.aml.name

target = databricks\_schema.silver.name # default database for unnamed tables

storage = "abfss://logs@${var.sa\_name}.dfs.core.windows.net/dlt/aml-core"

configuration = {

"pipelines.useCatalog" = "true"

}

library {

notebook { path = "/Repos/aml/dlt/transactions.py" }

}

continuous = true # or false for triggered

edition = "ADVANCED"

}

Start the pipeline once from the Databricks UI (or with databricks pipelines start --pipeline-id <id>). New files in raw/transactions/ will flow to **aml.raw.aml\_raw\_transactions → aml.silver.aml\_transactions → aml.gold.aml\_txn\_features**.

**Quick validation checklist**

* In **Data Explorer**, you should see **catalog aml** with schemas **raw/silver/gold**.
* Upload a test file to raw/transactions/ and watch DLT tables populate.
* SELECT \* FROM aml.silver.aml\_transactions LIMIT 10; returns rows.
* Privileges: engineers can CREATE TABLE in raw/silver, investigators can SELECT in gold.

**Next (Step 4 preview)**

* Add **sanctions enrichment** (OFAC/EU/SECO) into aml.silver.watchlists, and a first **baseline rules** DLT notebook (top 5 typologies).
* Wire **Power BI** to aml.gold via **Synapse Serverless SQL** or **Databricks SQL** (per your earlier choice).