**Step 4 = Enrichment + Baseline Rules**.  
You’ll (a) land/standardize sanctions lists, (b) conform parties/accounts, then (c) produce **gold.alerts** from 5 starter typologies. All inside Databricks (DLT), served to Power BI via Synapse Serverless.

**4A) Ingest & standardize sanctions lists (silver.watchlists)**

**What to drop in ADLS (raw):**

raw/watchlists/ofac/\*.csv

raw/watchlists/eu/\*.csv

raw/watchlists/seco/\*.csv

(Use ADF Copy → HTTP to fetch daily; schedule every 24h.)

**DLT notebook** /Repos/aml/dlt/watchlists.py

import dlt, re, unicodedata

from pyspark.sql.functions import col, upper, regexp\_replace, trim, to\_date, lit

ST\_RAW = "abfss://raw@<STORAGE>.dfs.core.windows.net/watchlists/"

CAT = "aml"; SCH\_SILVER = "silver"

def norm\_name(c):

if c is None: return None

x = unicodedata.normalize("NFKD", c)

x = "".join(ch for ch in x if not unicodedata.combining(ch))

x = re.sub(r"[^A-Za-z0-9 ]", " ", x).upper()

x = re.sub(r"\s+", " ", x).strip()

return x

@dlt.table(name="wl\_raw", table\_properties={"quality":"bronze"})

def wl\_raw():

return (spark.readStream

.format("cloudFiles").option("cloudFiles.format","csv").option("header","true")

.load(ST\_RAW)

.withColumn("source", regexp\_replace(col("\_metadata.file\_path"), r".\*/watchlists/([^/]+)/.\*", r"$1"))

)

@dlt.table(name=f"{CAT}.{SCH\_SILVER}.watchlists", table\_properties={"quality":"silver"})

def watchlists():

from pyspark.sql.functions import udf

norm = udf(norm\_name)

df = dlt.read\_stream("wl\_raw")

return (df

.withColumn("name\_norm", norm(trim(col("name"))))

.withColumn("aka\_norm", norm(trim(col("aka"))))

.withColumn("dob", to\_date("dob"))

.withColumn("country\_norm", upper(trim(col("country"))))

.select("source","name","aka","name\_norm","aka\_norm","dob","country","country\_norm","program","list\_id")

)

**4B) Conform parties & accounts (silver.\*)**

Assuming you already landed parties/accounts like transactions (Step 3). Add a small DLT to standardize:

@dlt.table(name="aml.silver.parties", table\_properties={"quality":"silver"})

def parties():

df = spark.read.table("aml.raw.parties\_raw") # or your raw name

return (df

.withColumn("name\_norm", upper(regexp\_replace(trim(col("name")), r"[^A-Za-z0-9 ]"," ")))

.withColumn("country\_norm", upper(trim(col("country"))))

.select("party\_id","name","name\_norm","dob","country","country\_norm","kyc\_risk\_rating")

)

@dlt.table(name="aml.silver.accounts", table\_properties={"quality":"silver"})

def accounts():

df = spark.read.table("aml.raw.accounts\_raw")

return df.select("account\_id","party\_id","iban","opened\_dt","status","country").dropDuplicates(["account\_id"])

**4C) Baseline rules (top 5 typologies) → gold.alerts**

Create /Repos/aml/dlt/rules\_baseline.py and add as a second library to your DLT pipeline.

**Config (set these in the pipeline “Configuration”)**

R\_SMALL\_TX=950.0

R\_STRUCTURING\_TOTAL=10000.0

R\_ROUND\_TRIP\_DAYS=5

R\_VELOCITY\_WINDOW\_DAYS=7

R\_VELOCITY\_BASELINE\_DAYS=30

RISK\_COUNTRIES=IR,AF,CU,KN,PA,YE,SO,LB,LY,SD,SY,IQ,UA,RU # example only

**Rules implementation**

import dlt, json

from pyspark.sql import functions as F, Window as W

# Helpers

def param(name, default):

return float(dlt.utils.get\_pipeline\_conf().get(name, default)) if str(default).replace('.','',1).isdigit() \

else dlt.utils.get\_pipeline\_conf().get(name, default)

SMALL = param("R\_SMALL\_TX", 950.0)

TOTAL = param("R\_STRUCTURING\_TOTAL", 10000.0)

RT\_DAYS = int(param("R\_ROUND\_TRIP\_DAYS", 5))

WIN\_D = int(param("R\_VELOCITY\_WINDOW\_DAYS", 7))

BASE\_D = int(param("R\_VELOCITY\_BASELINE\_DAYS", 30))

RISK\_C = set(str(param("RISK\_COUNTRIES","")).split(","))

tx = spark.read.table("aml.silver.aml\_transactions") # from Step 3

acct = spark.read.table("aml.silver.accounts")

party = spark.read.table("aml.silver.parties")

wl = spark.read.table("aml.silver.watchlists")

# R1 — Sanctions name/DOB/country match

@dlt.table(name="r1\_sanctions", table\_properties={"quality":"gold"})

def r1\_sanctions():

hits = (party.alias("p")

.join(wl.alias("w"),

(F.coalesce(F.col("p.name\_norm"),F.lit("")) == F.coalesce(F.col("w.name\_norm"),F.lit(""))) |

(F.coalesce(F.col("p.name\_norm"),F.lit("")) == F.coalesce(F.col("w.aka\_norm"),F.lit(""))), "inner")

.where( (F.col("p.dob").isNull()) | (F.col("p.dob")==F.col("w.dob")) )

.select(

F.col("p.party\_id").alias("subject\_id"),

F.lit("R1\_SANCTIONS\_MATCH").alias("typology"),

F.lit(0.99).alias("risk\_score"),

F.to\_json(F.struct("w.source","w.program","w.list\_id","p.name","w.name","w.aka")).alias("evidence")

)

)

return hits

# R2 — Structuring/smurfing: many small credits in 24h per account

@dlt.table(name="r2\_structuring", table\_properties={"quality":"gold"})

def r2\_structuring():

t = spark.read.table("aml.silver.aml\_transactions").filter(F.col("amount")>0) # credits

t = t.withColumn("d", F.to\_date("value\_date"))

w = W.partitionBy("account\_id","d")

agg = (t.filter(F.col("amount") <= F.lit(SMALL))

.groupBy("account\_id","d")

.agg(F.count("\*").alias("tx\_count"), F.sum("amount").alias("sum\_amt"))

.filter(F.col("sum\_amt") >= F.lit(TOTAL))

.select(

F.col("account\_id").alias("subject\_id"),

F.lit("R2\_STRUCTURING").alias("typology"),

F.lit(0.8).alias("risk\_score"),

F.to\_json(F.struct("tx\_count","sum\_amt","day","threshold\_small", "threshold\_total")

.withField("day", F.col("d"))

.withField("threshold\_small", F.lit(SMALL))

.withField("threshold\_total", F.lit(TOTAL))

).alias("evidence")

)

)

return agg

# R3 — High-risk corridors by country (origin/destination)

@dlt.table(name="r3\_high\_risk\_corridor", table\_properties={"quality":"gold"})

def r3\_high\_risk\_corridor():

rc = F.udf(lambda c: c in RISK\_C, "boolean")

return (tx

.where(rc(F.upper(F.col("beneficiary\_country"))) | rc(F.upper(F.col("origin\_country"))))

.select(

F.col("account\_id").alias("subject\_id"),

F.lit("R3\_HIGH\_RISK\_CORRIDOR").alias("typology"),

F.lit(0.7).alias("risk\_score"),

F.to\_json(F.struct("origin\_country","beneficiary\_country","tx\_id","amount")).alias("evidence")

)

)

# R4 — Velocity spike (7d vs 30d baseline)

@dlt.table(name="r4\_velocity\_spike", table\_properties={"quality":"gold"})

def r4\_velocity\_spike():

t = tx.select("account\_id","value\_date","amount").withColumn("d", F.to\_date("value\_date"))

win7 = W.partitionBy("account\_id").orderBy(F.col("d").cast("long")).rangeBetween(-WIN\_D\*86400, 0)

win30 = W.partitionBy("account\_id").orderBy(F.col("d").cast("long")).rangeBetween(-BASE\_D\*86400, 0)

s = (t

.withColumn("sum7", F.sum("amount").over(win7))

.withColumn("sum30", F.sum("amount").over(win30))

.withColumn("ratio", F.when(F.col("sum30")>0, F.col("sum7")/F.col("sum30")).otherwise(F.lit(0.0)))

.filter(F.col("ratio") >= 0.6) # tune

.select(

"account\_id",

F.lit("R4\_VELOCITY\_SPIKE").alias("typology"),

F.lit(0.6).alias("risk\_score"),

F.to\_json(F.struct("sum7","sum30","ratio")).alias("evidence")

).withColumnRenamed("account\_id","subject\_id")

)

return s

# R5 — Round-tripping: out & in between same pair, same amount within N days

@dlt.table(name="r5\_round\_tripping", table\_properties={"quality":"gold"})

def r5\_round\_tripping():

a = tx.select("tx\_id","value\_date","amount","account\_id","counterparty\_account\_id")

b = a.selectExpr("tx\_id as tx\_id2","value\_date as vd2","amount as amount2",

"counterparty\_account\_id as account\_id","account\_id as counterparty\_account\_id")

j = (a.join(b, (a.account\_id==b.account\_id) & (a.counterparty\_account\_id==b.counterparty\_account\_id) & (a.amount==b.amount2))

.where(F.abs(F.datediff(a.value\_date, b.vd2)) <= RT\_DAYS)

.select(

a.account\_id.alias("subject\_id"),

F.lit("R5\_ROUND\_TRIP").alias("typology"),

F.lit(0.65).alias("risk\_score"),

F.to\_json(F.struct(a.tx\_id.alias("tx\_out"), F.col("tx\_id2").alias("tx\_in"),

a.amount.alias("amount"), "counterparty\_account\_id")).alias("evidence")

).dropDuplicates()

)

return j

# Union → gold.alerts

@dlt.table(name="aml.gold.alerts", table\_properties={"quality":"gold"})

def alerts():

from functools import reduce

dfs = [dlt.read(t) for t in ["r1\_sanctions","r2\_structuring","r3\_high\_risk\_corridor","r4\_velocity\_spike","r5\_round\_tripping"]]

all\_alerts = reduce(lambda x,y: x.unionByName(y, allowMissingColumns=True), dfs)

return (all\_alerts

.withColumn("alert\_id", F.sha2(F.concat\_ws("|","subject\_id","typology","evidence"),256))

.withColumn("created\_ts", F.current\_timestamp())

.select("alert\_id","subject\_id","typology","risk\_score","evidence","created\_ts")

)

**Wire it in:** edit your existing DLT pipeline and **add another library**:

library {

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

}

library {

notebook { path = "/Repos/aml/dlt/rules\_baseline.py" }

}

Start/refresh the pipeline. You should now see **aml.gold.alerts** populating.

**4D) Expose gold.alerts to Power BI via Synapse Serverless SQL**

Create a **Serverless SQL database** and external objects using **Managed Identity**:

-- In Synapse Serverless

CREATE DATABASE aml\_bi;

GO

USE aml\_bi;

GO

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'A\_Strong\_Temp\_Password\_Only\_Used\_Here';

CREATE DATABASE SCOPED CREDENTIAL msi\_cred WITH IDENTITY = 'Managed Identity';

CREATE EXTERNAL DATA SOURCE ds\_gold

WITH ( TYPE = HADOOP, LOCATION = 'abfss://gold@<STORAGE>.dfs.core.windows.net', CREDENTIAL = msi\_cred );

-- Query Delta directly (no external table required):

SELECT \*

FROM OPENROWSET(

BULK 'alerts', -- folder under gold/

DATA\_SOURCE = 'ds\_gold',

FORMAT = 'DELTA'

) AS rows

WHERE created\_ts >= DATEADD(day, -7, SYSUTCDATETIME());

Now point **Power BI** to your Synapse Serverless endpoint and use the query above (or create a view on top of it for convenience).

**Run-order checklist**

1. ADF Copy jobs drop sanctions files into raw/watchlists/....
2. DLT pipeline (continuous/triggered) runs:
   * watchlists.py → aml.silver.watchlists
   * rules\_baseline.py → aml.gold.alerts
3. Synapse Serverless SQL reads gold/alerts → Power BI dashboards (investigator queue, typologies, trends).