End-to-End Monthly AWS Cost ETL Pipeline using Databricks and DBFS

# Overview

You will build a data pipeline with these stages:  
  
1. Raw (Landing Zone): Monthly cost data pulled from AWS via boto3 and saved as JSON.  
2. Bronze (Ingest): Auto Loader ingests new files into a Delta Lake table.  
3. Silver (Transform): Deduplicated and cleaned version of the bronze table.  
4. Gold (Aggregate - optional): Monthly service cost summary for reporting.  
  
All stages will be stored in Databricks File System (DBFS).

# Step 0: Setup DBFS Directories

Mount S3 (if needed):

dbutils.fs.mount(  
 source = "s3a://your-bucket-name",  
 mount\_point = "/mnt/your-bucket-name",  
 extra\_configs = {"fs.s3a.aws.credentials.provider": "com.amazonaws.auth.DefaultAWSCredentialsProviderChain"}  
)

Create Required Directories in DBFS:

dbutils.fs.mkdirs("/mnt/raw/aws\_cost/")  
dbutils.fs.mkdirs("/mnt/bronze/aws\_cost/")  
dbutils.fs.mkdirs("/mnt/silver/aws\_cost/")  
dbutils.fs.mkdirs("/mnt/gold/aws\_cost\_summary/")  
dbutils.fs.mkdirs("/mnt/checkpoints/aws\_cost\_bronze/")

# Step 1: Ingest Monthly AWS Cost via Boto3

This script runs once per month and dumps AWS cost data into DBFS in JSON format:

import boto3  
import pandas as pd  
from datetime import date, timedelta  
  
client = boto3.client("ce", region\_name="us-east-1")  
  
today = date.today()  
first\_of\_this\_month = today.replace(day=1)  
first\_of\_last\_month = (first\_of\_this\_month - timedelta(days=1)).replace(day=1)  
  
start = first\_of\_last\_month.strftime('%Y-%m-%d')  
end = first\_of\_this\_month.strftime('%Y-%m-%d')  
  
response = client.get\_cost\_and\_usage(  
 TimePeriod={"Start": start, "End": end},  
 Granularity="MONTHLY",  
 Metrics=["AmortizedCost"],  
 GroupBy=[{"Type": "DIMENSION", "Key": "SERVICE"}]  
)  
  
records = []  
for group in response["ResultsByTime"][0]["Groups"]:  
 records.append({  
 "month": start,  
 "service": group["Keys"][0],  
 "amount": float(group["Metrics"]["AmortizedCost"]["Amount"]),  
 "unit": group["Metrics"]["AmortizedCost"]["Unit"]  
 })  
  
df = pd.DataFrame(records)  
output\_path = f"/dbfs/mnt/raw/aws\_cost/aws\_cost\_{start}.json"  
df.to\_json(output\_path, orient="records", lines=True)

# Step 2: Auto Loader to Ingest JSON to Bronze Table

from pyspark.sql.types import StructType, StringType, DoubleType  
  
schema = StructType() \  
 .add("month", StringType()) \  
 .add("service", StringType()) \  
 .add("amount", DoubleType()) \  
 .add("unit", StringType())  
  
df = (spark.readStream  
 .format("cloudFiles")  
 .option("cloudFiles.format", "json")  
 .schema(schema)  
 .load("/mnt/raw/aws\_cost/")  
)  
  
(df.writeStream  
 .format("delta")  
 .outputMode("append")  
 .option("checkpointLocation", "/mnt/checkpoints/aws\_cost\_bronze/")  
 .start("/mnt/bronze/aws\_cost/")  
)

# Step 3: Silver Layer – Clean & Deduplicate

from pyspark.sql.functions import row\_number  
from pyspark.sql.window import Window  
  
bronze\_df = spark.read.format("delta").load("/mnt/bronze/aws\_cost/")  
  
window\_spec = Window.partitionBy("month", "service").orderBy("amount".desc())  
  
deduped\_df = bronze\_df.withColumn("row\_num", row\_number().over(window\_spec)) \  
 .filter("row\_num = 1") \  
 .drop("row\_num")  
  
deduped\_df.write.format("delta").mode("overwrite").save("/mnt/silver/aws\_cost/")

# Step 4 (Optional): Gold Layer – Monthly Aggregation

silver\_df = spark.read.format("delta").load("/mnt/silver/aws\_cost/")  
  
monthly\_summary = silver\_df.groupBy("month", "service") \  
 .sum("amount") \  
 .withColumnRenamed("sum(amount)", "monthly\_cost")  
  
monthly\_summary.write.format("delta").mode("overwrite").save("/mnt/gold/aws\_cost\_summary/")

# Scheduling in Databricks

You can use Databricks Workflows to schedule the following:  
- Notebook 1: Monthly boto3 pull (Step 1)  
- Notebook 2: Bronze Auto Loader (Step 2) – continuously or triggered  
- Notebook 3: Silver transform (Step 3)  
- Notebook 4 (Optional): Gold aggregation (Step 4)