Databricks ETL Pipeline & Interactive Dashboard Queries with Visualizations

In this document, we will describe how to automate an ETL pipeline in Databricks that pulls AWS cost data using the AWS Cost Explorer API with Boto3, processes it, stores it in a Delta table, and generates interactive queries for a dashboard. Along with the SQL queries, we will provide suggestions for visualizations, including what should be on the x-axis, y-axis, and in 'Group By' for each query. These queries and visualizations will help businesses monitor their cloud spend, identify inefficiencies, and make data-driven decisions on resource allocation.

# 1. ETL Pipeline Setup

The goal of the ETL pipeline is to pull daily cost data from AWS Cost Explorer API using Boto3, process the data, and store it in a Delta table in Databricks. Below is the Python code to set up the process.

### Prerequisites:  
1. Create a Databricks secret scope called `aws` and add `access\_key` and `secret\_key`.  
2. Have an IAM role (`DatabricksCostRole`) in AWS that can access the Cost Explorer API and assume the role from Databricks.

### Code to Automate ETL Pipeline:  
```python  
# %pip install boto3  
import boto3  
import datetime  
from pyspark.sql import SparkSession  
  
# Load base credentials from Databricks secrets  
ACCESS\_KEY = dbutils.secrets.get(scope='aws', key='access\_key')  
SECRET\_KEY = dbutils.secrets.get(scope='aws', key='secret\_key')  
  
# Assume Role for temporary credentials  
sts\_client = boto3.client(  
 'sts',  
 aws\_access\_key\_id=ACCESS\_KEY,  
 aws\_secret\_access\_key=SECRET\_KEY  
)  
  
role\_arn = 'arn:aws:iam::<your\_account\_id>:role/DatabricksCostRole'  
assumed\_role = sts\_client.assume\_role(  
 RoleArn=role\_arn,  
 RoleSessionName='databricks-cost-session'  
)  
  
temp\_credentials = assumed\_role['Credentials']  
  
# Initialize AWS Cost Explorer client  
ce\_client = boto3.client(  
 'ce',  
 region\_name='us-east-1',  
 aws\_access\_key\_id=temp\_credentials['AccessKeyId'],  
 aws\_secret\_access\_key=temp\_credentials['SecretAccessKey'],  
 aws\_session\_token=temp\_credentials['SessionToken']  
)  
  
# Define date range (Jan 1st of last year to today)  
today = datetime.date.today()  
start\_date = datetime.date(today.year - 1, 1, 1)  
  
def get\_month\_ranges(start, end):  
 ranges = []  
 current = start  
 while current < end:  
 next\_month = (current.replace(day=28) + datetime.timedelta(days=4)).replace(day=1)  
 ranges.append((current, min(next\_month, end)))  
 current = next\_month  
 return ranges  
  
# Pull data from AWS Cost Explorer  
all\_rows = []  
  
for start, end in get\_month\_ranges(start\_date, today):  
 response = ce\_client.get\_cost\_and\_usage(  
 TimePeriod={'Start': start.strftime('%Y-%m-%d'), 'End': end.strftime('%Y-%m-%d')},  
 Granularity='DAILY',  
 Metrics=['AmortizedCost'],  
 GroupBy=[{'Type': 'DIMENSION', 'Key': 'SERVICE'}]  
 )  
  
 for day in response['ResultsByTime']:  
 for group in day['Groups']:  
 service = group['Keys'][0]  
 amount = float(group['Metrics']['AmortizedCost']['Amount'])  
 all\_rows.append((day['TimePeriod']['Start'], service, amount))  
  
# Convert to Spark DataFrame  
spark = SparkSession.builder.getOrCreate()  
  
df = spark.createDataFrame(all\_rows, ['date', 'service', 'amount'])  
  
# Save as Delta Table  
df.write.mode('overwrite').format('delta').saveAsTable('aws\_costs.daily\_spent')  
```

# 2. Business Queries for the Dashboard

Once the data is processed and stored in the Delta table, you can use the following SQL queries to generate interactive dashboards for monitoring and optimizing AWS cloud costs. These queries help answer key business questions such as how much the business is spending on each service, identifying cost spikes, and understanding monthly trends. The document also suggests the best visualization types, with details for the x-axis, y-axis, and grouping.

## Total Cost (Overall Spend)

SELECT ROUND(SUM(amount), 2) AS total\_spent FROM aws\_costs.daily\_spent;

Suggested Visualization: Visualization: Single Value Visualization  
X-Axis: N/A  
Y-Axis: Total Cost  
Group By: N/A

## Monthly Spend Trend (Cost Trend Over Time)

SELECT DATE\_TRUNC('month', date) AS month, ROUND(SUM(amount), 2) AS total\_spent FROM aws\_costs.daily\_spent GROUP BY month ORDER BY month;

Suggested Visualization: Visualization: Line Chart  
X-Axis: Month  
Y-Axis: Total Spend  
Group By: N/A

## Top 10 Services by Total Spend

SELECT service, ROUND(SUM(amount), 2) AS total\_spent FROM aws\_costs.daily\_spent GROUP BY service ORDER BY total\_spent DESC LIMIT 10;

Suggested Visualization: Visualization: Bar Chart  
X-Axis: Service  
Y-Axis: Total Spend  
Group By: N/A

## Monthly Cost Breakdown by Service

SELECT DATE\_TRUNC('month', date) AS month, service, ROUND(SUM(amount), 2) AS total\_spent FROM aws\_costs.daily\_spent GROUP BY month, service ORDER BY month, total\_spent DESC;

Suggested Visualization: Visualization: Stacked Bar Chart  
X-Axis: Month  
Y-Axis: Total Spend  
Group By: Service

## Daily Cost Summary

SELECT date, ROUND(SUM(amount), 2) AS total\_spent FROM aws\_costs.daily\_spent GROUP BY date ORDER BY date;

Suggested Visualization: Visualization: Line Chart  
X-Axis: Date  
Y-Axis: Total Spend  
Group By: N/A

## Service Share of Total Cost (%)

SELECT service, ROUND(100 \* SUM(amount) / SUM(SUM(amount)) OVER (), 2) AS percent\_share FROM aws\_costs.daily\_spent GROUP BY service ORDER BY percent\_share DESC;

Suggested Visualization: Visualization: Pie Chart  
X-Axis: N/A  
Y-Axis: Percent Share  
Group By: Service

## Peak Day Spend (Identify Maximum Daily Spend)

SELECT date, ROUND(SUM(amount), 2) AS total\_spent FROM aws\_costs.daily\_spent GROUP BY date ORDER BY total\_spent DESC LIMIT 1;

Suggested Visualization: Visualization: Single Value Visualization  
X-Axis: N/A  
Y-Axis: Total Cost  
Group By: N/A

## Cost Anomalies (Identify Unusual Spikes)

SELECT date, ROUND(SUM(amount), 2) AS total\_spent FROM aws\_costs.daily\_spent GROUP BY date HAVING total\_spent > 2 \* (SELECT AVG(amount) FROM aws\_costs.daily\_spent) ORDER BY total\_spent DESC;

Suggested Visualization: Visualization: Bar Chart  
X-Axis: Date  
Y-Axis: Total Spend  
Group By: N/A

## Rolling 7-Day Average Spend (Moving Average)

SELECT date, ROUND(SUM(amount), 2) AS daily\_spent, ROUND(AVG(SUM(amount)) OVER (ORDER BY date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW), 2) AS moving\_avg\_7d FROM aws\_costs.daily\_spent GROUP BY date ORDER BY date;

Suggested Visualization: Visualization: Line Chart  
X-Axis: Date  
Y-Axis: Spend  
Group By: N/A

## Year-over-Year Spend Comparison (Cost Trend Analysis)

SELECT EXTRACT(YEAR FROM date) AS year, EXTRACT(MONTH FROM date) AS month, ROUND(SUM(amount), 2) AS total\_spent FROM aws\_costs.daily\_spent GROUP BY year, month ORDER BY year, month;

Suggested Visualization: Visualization: Bar Chart  
X-Axis: Month  
Y-Axis: Total Spend  
Group By: Year