AWS Cost and Usage Data Pull in Databricks

This document provides a step-by-step guide to pulling AWS Cost and Usage data into Databricks notebooks using the AWS Cost Explorer API and Boto3 library.

# Step 1: Install Required Libraries

First, install the necessary libraries. We will use `boto3` to interact with AWS services (Cost Explorer API).

Install `boto3` library in Databricks cluster using the following command:

%pip install boto3

# Step 2: Set Up Authentication with AWS

You need to authenticate using your AWS credentials. Store your credentials securely in Databricks Secrets.

## Store AWS Credentials in Databricks Secrets

Store your AWS Access Key and AWS Secret Access Key in Databricks Secrets Manager to avoid hardcoding credentials.

# Retrieve AWS credentials securely from Databricks Secrets  
aws\_access\_key\_id = dbutils.secrets.get(scope="aws-secrets", key="aws-access-key-id")  
aws\_secret\_access\_key = dbutils.secrets.get(scope="aws-secrets", key="aws-secret-access-key")

## Initialize Boto3 Client

Create the Boto3 client for interacting with AWS Cost Explorer.

import boto3  
  
# Initialize the AWS Cost Explorer client with credentials  
client = boto3.client(  
 'ce', # AWS Cost Explorer service  
 aws\_access\_key\_id=aws\_access\_key\_id,  
 aws\_secret\_access\_key=aws\_secret\_access\_key,  
 region\_name='us-east-1' # Cost Explorer is available in the 'us-east-1' region  
)

# Step 3: Define Parameters for the API Call

Define the time period, granularity, and metrics for the GetCostAndUsage API call. Adjust values based on your requirements.

# Define time period (e.g., March 2025)  
time\_period = {  
 'Start': '2025-03-01', # Start date  
 'End': '2025-03-31' # End date  
}  
  
# Set granularity (DAILY or MONTHLY)  
granularity = 'DAILY'  
  
# Define the metrics you want (e.g., 'BlendedCost')  
metrics = ['BlendedCost']  
  
# Optionally define a filter (e.g., by service, such as EC2 or S3)  
filter = {  
 'Dimensions': {  
 'Key': 'SERVICE',  
 'Values': ['Amazon Elastic Compute Cloud - Compute'] # Filter by EC2 service  
 }  
}

# Step 4: Call the AWS Cost Explorer API

Now, make the actual API call to GetCostAndUsage using the parameters you have defined.

# Make the API call to get cost and usage data for the defined time period  
response = client.get\_cost\_and\_usage(  
 TimePeriod=time\_period, # Define the time period  
 Granularity=granularity, # Define the granularity (daily or monthly)  
 Metrics=metrics, # Define the cost metrics (e.g., BlendedCost)  
 Filter=filter # Optional filter to narrow down the data  
)

# Step 5: Process the API Response

After receiving the API response, process the data to extract the relevant information and structure it for analysis.

# Extract the cost and usage data from the API response  
results = response['ResultsByTime']  
data = []  
  
# Iterate through the results and extract date, service, and cost information  
for result in results:  
 for group in result.get('Groups', []):  
 # Extract the cost amount and service name  
 amount = group['Metrics']['BlendedCost']['Amount']  
 service = group['Keys'][0] # The service name (e.g., EC2)  
 date = result['TimePeriod']['Start'] # Date of the cost data  
 data.append((date, service, float(amount)))  
  
# Convert the data into a Spark DataFrame for further analysis  
from pyspark.sql import SparkSession  
from pyspark.sql.types import StructType, StructField, StringType, DoubleType  
  
# Define the schema for the DataFrame  
schema = StructType([  
 StructField('date', StringType(), True),  
 StructField('service', StringType(), True),  
 StructField('blended\_cost', DoubleType(), True)  
])  
  
# Create the DataFrame  
spark = SparkSession.builder.getOrCreate()  
df = spark.createDataFrame(data, schema=schema)  
  
# Display the DataFrame  
display(df)

# Step 6: Analyze the Data

Now that the data is in a Spark DataFrame, you can use Databricks features to analyze, visualize, or store the data.

# Group by service and calculate total blended cost  
df\_grouped = df.groupBy("service").sum("blended\_cost").withColumnRenamed("sum(blended\_cost)", "total\_cost")  
  
# Display the grouped data  
display(df\_grouped)