**A diagram of a company

AI-generated content may be incorrect.**

## Solution Overview:

1. CloudWatch event rule is scheduled to trigger the lambda function depending on the configuration, this could be set to run daily, weekly, or monthly depending upon the variable you set.
2. Once triggered, the lambda function is going to execute a python script which is designed to fetch and store cost usage data.
3. The script initialized aws SDK clients to communicate with both Cost Explorer API and S3 bucket. It looks for two environment variables:
   1. CUR\_BUCKET: the target S3 bucket where the report will be stored.
   2. CUR\_RANGE: determines whether the report should include cost data for a day, a week, or a month
4. Based on the selected CUR\_RANGE, it pulls the relevant cost data from AWS Cost Explorer, formats it into a CSV file, and uploads it to the specified S3 bucket.
5. Date Range logic:
   1. **Daily**: If run on March 21, 2025, the report will include costs for March 19, 2025.
   2. **Weekly**: If run on March 21, 2025, the report will cover March 13–19, 2025.
   3. **Monthly**: If run on March 2, 2025, the report will cover February 1–28, 2025.
6. The Lambd function assumes an IAM role with the necessary permissions to cal the cost explorer api and write to the s3 bucket.
7. Execution logs of the lambda function are stored in cloudwatch with 14 days retention
8. AWS budget will trigger an alert whenever the cost reaches the threshold of 80% of the cost limit ($1000 – set currently)

## Reasoning Behind the Architecture

1. The goal is to build a flexible and cost-efficient system that still provides complete cost visibility on a daily, weekly, or monthly basis. Real-time monitoring wasn’t practical because AWS Cost Explorer doesn’t update cost data immediately—it can take up to 24 hours. Trying to fetch cost data more frequently would be misleading and unnecessarily expensive due to repeated Lambda invocations and S3 writes.
2. The solution is built entirely using AWS-native services, making the integration seamless and reducing the overhead of managing third-party tools.
3. AWS Lambda is a great fit here because the task is lightweight, scheduled, and doesn’t require persistent infrastructure. It's easy to manage and scales automatically.
4. Being serverless, Lambda follows a pay-as-you-go model, making it a cost-effective choice for periodic jobs like this.
5. CloudWatch logging provides operational visibility. To keep logging costs in check, log retention is capped at 14 days.
6. The S3 bucket used for storing reports is restricted through a tight bucket policy (limited to the same AWS account) and is configured with **lifecycle-configuration**, which helps optimize storage costs without compromising access speed.

## Cost Estimation

|  |  |
| --- | --- |
| **Service** | **Cost** |
| Lambda | $0.00 |
| S3 | $0.023 |
| EventBridge | $0.00 |
| Budgets | $0.00 |
| Total | $0.023 |

Rationale: Cost is calculated using AWS pricing Calculator <https://calculator.aws/#/>

1. Lambda – 128mb for 40 secs = $0  
   Lambda charges $0 for 1 million requests per month and 400000 GB-s computes

A screenshot of a calculator

AI-generated content may be incorrect.

1. S3: The size of file can vary based on number of rows in CSV. It could be in KB, MB or in GB. Assuming it is in MB. S3 charges $0.023 per GB for first 50TB/ month. Also, S3 offer 2000 free PUT requests, so no cost for those if we are just sending it once a day i.e. total 30 PUT request.
2. **Cost Explorer:** Cost Explorer API is free for monthly and daily granularity.
3. **EventBridge**: You can make 14,000,000 invocations per month for free. Then $1.00/million scheduled invocations per month. (For above scenario, free tier will suffice)