International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 4, July- August 2024 |

DOI:10.15662/IJEETR.2024.0604011

An Automated, Low-Cost AWS Cost Monitoring Framework for Daily Operational Anomaly Detection

Bhanuprakash Suravarapu

DevOps Engineer, Amazon Web Services Inc., USA

bhanuprakash.suravarapu6@gmail.com

Praneeth Ganta

Senior Software Engineer, LinkedIn, USA

praneethganta@gmail.com

Tulasi priya Vattikuti

Cloud Data Engineer, Medtronic, USA

tulcpriya@gmail.com

Vasu Babu Narra

Senior Release Manager, Uipath Inc., USA

narravasu77@gmail.com

ABSTRACT: Cloud workloads deployed on elastic infrastructure frequently experience cost anomalies caused by scaling misconfigurations, ingestion failures, or idle resource consumption. Such anomalies often remain unnoticed until billing cycles are complete, causing financial and operational risk. Commercial cost monitoring tools provide advanced analytics but introduce licensing and operational complexity unsuitable for many teams. This paper presents an automated AWS cost monitoring framework that retrieves billing data, performs daily trend analysis, and distributes anomaly-aware cost summaries using native AWS services. Implemented using serverless components and infrastructure-as-code automation, the system operates with minimal maintenance overhead while providing actionable daily cost visibility. Production usage demonstrates improved operational awareness and early detection of workload issues without reliance on third-party platforms.

KEYWORDS: Cloud cost monitoring, anomaly detection, AWS Lambda, serverless automation, FinOps

I. INTRODUCTION

Cloud computing enables rapid deployment of scalable workloads but introduces challenges in cost visibility. Elastic scaling and service integration can cause cost variations that remain unnoticed when billing inspection occurs only monthly. Failures in ingestion pipelines, incorrect auto-scaling policies, or idle clusters may continue for extended periods without operational detection. Organizations often rely on manual inspection or costly third-party monitoring solutions.

This research presents a lightweight automated framework that provides daily cost summaries and anomaly indicators using native AWS services with minimal operational overhead.

II. PROBLEM STATEMENT

Modern data workloads such as Apache Flink pipelines, EMR clusters, and analytics services consume resources across multiple AWS services, including compute, storage, data transfer, and monitoring components.
Cost anomalies typically manifest as:

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 4, July- August 2024 |

DOI:10.15662/IJEETR.2024.0604011

- Sudden cost increases due to scaling misconfiguration
- Sudden decreases caused by ingestion failures
- Gradual increases due to resource leaks
- Persistent idle resources are consuming compute capacity

Detecting such changes daily enables faster remediation and prevents long-term operational impact.

III. SYSTEM ARCHITECTURE

A. Architectural Goals

The framework design prioritizes:

- Minimal maintenance overhead
- Automated deployment
- Low execution cost
- Cross-account deployability
- Daily anomaly visibility

B. Architecture Overview

The framework uses native AWS services:

- **AWS Lambda** – processing logic
- **AWS Cost Explorer API** – billing data retrieval
- **Amazon EventBridge** – daily scheduling
- **Amazon SNS** – report distribution
- **Terraform** – infrastructure deployment

Execution pipeline:

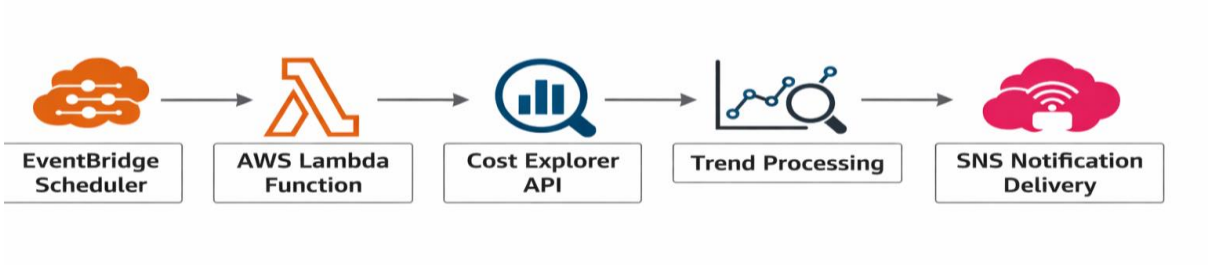

Fig 1: Architecture diagram of the workflow

C. System Operation Overview

The architecture illustrated in Fig. X presents a linear, event-driven workflow for automated daily cost monitoring. The system operates using native AWS services and executes without persistent infrastructure components.

The process begins with a scheduled rule in Amazon EventBridge, which triggers the AWS Lambda function once per day according to a defined cron expression. This scheduling mechanism ensures deterministic daily execution without manual intervention.

Upon invocation, the Lambda function queries the AWS Cost Explorer API to retrieve cost and usage data at daily granularity. The request specifies the time window and retrieves unblended cost metrics, which reflect actual usage charges rather than amortized or blended values. The response contains structured cost data grouped by date.

The retrieved cost data is then processed within the Lambda execution environment. The processing stage performs the following operations:

- Reorganizes cost records by account and date.
- Computes day-over-day cost differences.
- Calculates percentage deltas using:

$\Delta t = Ct - Ct-1Ct-1 \backslash Delta\_t = \backslash frac\{C\_t - C\_\{t-1\}\}\{C\_\{t-1\}\}\Delta t = Ct - Ct-1Ct-1Ct - Ct-1$

where CtC_tCt represents the cost at time.

- Classifies deltas into predefined anomaly categories based on static threshold values.

This transformation converts raw billing data into structured anomaly-aware output.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 4, July- August 2024 |

DOI:10.15662/IJEETR.2024.0604011

After processing, the formatted report is constructed as a fixed-width plain-text summary to ensure compatibility across email clients. Accounts with zero cost values are filtered to improve readability and reduce noise.
Finally, the report is delivered via Amazon Simple Notification Service (SNS). The Lambda function publishes the generated message to a configured SNS topic, which distributes notifications to subscribed recipients. This mechanism decouples processing logic from delivery endpoints and allows flexible subscription management.
The system does not maintain a persistent state between executions. All cost retrieval and anomaly computation are performed during each invocation based solely on Cost Explorer data. Infrastructure components—including the Lambda function, IAM roles, EventBridge schedule, and SNS topics—are provisioned using Terraform to ensure reproducible deployment.
This architecture achieves automated daily cost visibility with minimal operational overhead, leveraging managed cloud services and avoiding dedicated servers or external monitoring platforms.

Daily execution performs:
- Retrieve billing data per account.
- Reorganize costs by date.
- Compute day-over-day cost deltas.
- Aggregate totals.
- Generate a formatted report.
- Deliver the report via SNS email.

Accounts with zero spend are filtered to improve report readability.

IV. IMPLEMENTATION DETAILS

This section presents **core implementation logic** along with an explanation.

A. Lambda Execution Pipeline

The Lambda handler coordinates data retrieval, processing, and reporting:

```
def lambda_handler(context=None, event=None):
  mainCostDict = ce_get_costinfo_per_account(accountDict)
  mainDailyDict = process_costchanges_per_day(mainCostDict)
  mainDisplayDict = process_costchanges_for_display(mainDailyDict)
  finalDisplayDict= process_percentchanges_per_day(mainDisplayDict)
  send_report_sns(None, finalDisplayDict)
```

Description

Execution stages:
- Retrieve billing data.
- Reorganize cost structure by date.
- Prepare display-ready structure.
- Compute cost changes.
- Publish notification.

This pipeline ensures the separation of data acquisition and formatting logic.

B. Cost Retrieval Logic

Daily cost data is retrieved using AWS Cost Explorer:

```
response = cost_explorer.get_cost_and_usage(
    TimePeriod={
      'Start': START_DATE,
      'End': END_DATE
      },
  Granularity='DAILY',
  Metrics=['UnblendedCost']
)
```

Description
- Daily granularity aligns with Cost Explorer updates.
- Unblended cost reflects raw consumption.
- The retrieval window includes multiple days for comparison.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 4, July- August 2024 |

DOI:10.15662/IJEETR.2024.0604011

Optional tag filtering was initially implemented, but later made optional to avoid incomplete reporting when tagging is inconsistent.

C. Daily Delta Computation

Cost changes are computed per account:

```
percentDelta = (current_cost - previous_cost) / previous_cost
```

Handling Considerations

- The first day has no previous comparison.
- Division by zero is handled when the previous cost equals zero.
- Missing data entries are ignored.

This computation provides anomaly signals rather than forecasting.

D. Delta Classification Logic

Cost changes are translated into visual indicators:

```
def format_delta(delta):
    if delta > 0.15:
        return "↑↑"
    elif delta > 0.05:
        return "↑"
    elif delta < -0.15:
        return "↓↓"
    elif delta < -0.05:
        return "↓"
    return ""
```

E. Report Formatting Logic

Plain-text formatting ensures compatibility across email clients:

```
text_report += (
    f"{reportDate} | {account_name:10s} | "
    f"{cost_str} | {delta_str}\n"
)
```

Example output:

```
Date       | Account | Cost       | Change
2026-02-12 | prod    | $95.17     | 0.02%
2026-02-13 | prod    | $53.14     | -44% ↓↓
```

F. SNS Notification Publishing

Reports are delivered via SNS:

```
sns_client.publish(
    TopicArn=SNS_TOPIC_ARN,
    Subject="AWS Cost Report",
    Message=text_message
)
```

SNS simplifies notification delivery by avoiding domain verification and additional mail configuration.

G. Infrastructure Deployment

Deployment is automated via Terraform:

```
org_name        = "company"
team_name       = "analytics"
environment     = "prod"
recipient_emails = ["team@example.com"]
report_schedule  = "cron(30 11 * * ? *)"
```

Infrastructure modules define:

- IAM roles
- Lambda deployment

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 4, July- August 2024 |

DOI:10.15662/IJEETR.2024.0604011

- Scheduling configuration
- SNS topics

This enables rapid deployment across environments.

V. OPERATIONAL USE CASES

A. Streaming Pipeline Failure Detection

- A streaming pipeline exhibited a sudden cost reduction. Investigation revealed ingestion failure, causing compute resources to idle. Daily reporting enabled faster remediation.
- Daily monitoring revealed persistently elevated cost levels due to clusters remaining active beyond their intended lifetime. Monitoring enabled the shutdown of unused resources.
- Unexpected scaling policies caused compute expansion. Daily anomaly reporting exposed the issue for correction.

VI. EVALUATION OBSERVATIONS

Deployment usage demonstrates:

- Daily automated cost visibility
- Detection of unexpected workload behavior
- Reduced manual billing inspection

This paper does not claim quantitative benchmarking results beyond observed operational benefits.

VII. FUTURE WORK

Potential future extensions include:

- Slack/Teams alert integration
- Threshold-based alerts
- Historical cost visualization
- Predictive anomaly detection
- Multi-cloud monitoring support

VIII. CONCLUSION

This paper presents an automated framework for daily AWS cost monitoring designed to minimize operational overhead while improving cost visibility. By leveraging serverless architecture and infrastructure automation, organizations can achieve effective anomaly detection without adopting expensive monitoring platforms.

REFERENCES

[1] Amazon Web Services, “AWS Cost Explorer User Guide.” [Online]. Available: https://docs.aws.amazon.com/cost-management/latest/userguide/ce-what-is.html

[2] Amazon Web Services, “AWS Lambda Developer Guide.” [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

[3] Amazon Web Services, “Amazon EventBridge User Guide.” [Online]. Available: https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-what-is.html

[4] Amazon Web Services, “Amazon Simple Notification Service (SNS) Developer Guide.” [Online]. Available: https://docs.aws.amazon.com/sns/latest/dg/welcome.html

[5] HashiCorp, “Terraform Documentation.” [Online]. Available: https://developer.hashicorp.com/terraform/docs

[6] HashiCorp, “Terraform S3 Backend Configuration.” [Online]. Available: https://developer.hashicorp.com/terraform/language/settings/backends/s3

[7] FinOps Foundation, “FinOps Framework.” [Online]. Available: https://www.finops.org/framework/

[8] Amazon Web Services, “AWS Well-Architected Framework – Cost Optimization Pillar.” [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/cost-optimization-pillar/welcome.html