Technical Documentation: Automated AWS Cost Calculator

Document Version: 1.0
Author: Cloud Engineer
Date: October 15, 2025

1.0 Introduction

This document provides a detailed technical overview of the Automated AWS Cost Calculator system. Its purpose is to serve as a guide for engineers responsible for maintaining, extending, or troubleshooting the application and its underlying infrastructure.

The system is a serverless web application designed to provide AWS account cost visibility through a dynamic frontend dashboard, proactive budget alerting, and scheduled email reports. The entire stack is managed via Infrastructure as Code (Terraform) and deployed through a secure, automated CI/CD pipeline in GitHub Actions.

2.0 System Architecture

The architecture is logically separated into three main components: a serverless backend API, a static frontend application, and an event-driven alerting/reporting system.

2.1 High-Level Component Diagram

code Mermaid
downloadcontent_copy
expand_less

```
   graph TD
    subgraph "CI/CD Pipeline (GitHub Actions)"
        direction LR
        A[Git Push/PR] --> B[Static Analysis (TFLint, Checkov)];
        B --> C[Terraform Plan/Apply];
        C --> D[Infrastructure Provisioning];
        D --> E[Frontend Deployment];
    end
```

```
    subgraph "AWS Environment"
        U[Client Browser] --> CF[CloudFront Distribution];
        CF --> S3[S3 Bucket (Private Origin)];

        subgraph "API Path"
            CF -- "/costs" --> APIGW[API Gateway (HTTP API)];
            APIGW -- "Proxy Integration" --> LambdaAPI[GetCostDataApi
Lambda];
            LambdaAPI -- "ce:GetCostAndUsage" --> CE[AWS Cost
Explorer];
        end

        subgraph "Alerting Path"
            CWAlarm[CloudWatch Billing Alarm] -- "ALARM State" -->
SNSTopic[SNS Topic];
            SNSTopic --> SESSub[SES Email Subscription];
        end

        subgraph "Reporting Path"
            CWEvent[CloudWatch Event Rule (Cron)] -- "Scheduled
Trigger" --> LambdaReport[GetWeeklyCostReport Lambda];
            LambdaReport -- "ses:SendEmail" --> SES[SES Service];
            LambdaReport -- "sqs:SendMessage" --> DLQ[SQS Dead Letter
Queue];
        end

        subgraph "Security & Observability"
            KMS[KMS CMKs] -- "Encrypts" --> LambdaEnv[Lambda Env
Vars];
            KMS -- "Encrypts" --> SQSQueue[SQS Queues];
            KMS -- "Encrypts" --> CWLogs[CloudWatch Logs];
            LambdaAPI -- "Sends Traces" --> XRay[AWS X-Ray];
            LambdaReport -- "Sends Traces" --> XRay;
        end
    end
```

2.2 Frontend Architecture

- **Hosting:** The frontend is a static single-page application (HTML, CSS, JS) hosted in a private Amazon S3 bucket.
- **Content Delivery:** An Amazon CloudFront distribution serves as the public entry point. It provides a secure HTTPS endpoint, global caching for low latency, and a secure access layer to the private S3 origin.
- **Security:** Access to the S3 bucket is restricted exclusively to the CloudFront distribution via an **Origin Access Identity (OAI)**. All direct public access to the S3 bucket is blocked. The CloudFront distribution enforces a TLSv1.2_2021 security policy.

2.3 Serverless API Architecture

- **Endpoint:** A version 2 HTTP API Gateway provides the public endpoint (/costs). It is configured with a permissive CORS policy to allow GET requests from any origin.
- **Integration:** The /costs route uses a Lambda Proxy Integration, which forwards the raw request to the GetCostDataApi Lambda function and returns its response directly.
- **Business Logic:** The GetCostDataApi Lambda function contains the logic to query the AWS Cost Explorer API for the last 7 days of cost data, grouped by service.
- **Data Transformation:** The function processes the raw Cost Explorer data, transforms it into a clean JSON structure, and enriches it with user-friendly service names before returning it.
- **Observability:** API Gateway access logs are enabled and stored in an encrypted CloudWatch Log Group. The Lambda function has AWS X-Ray active tracing enabled.

2.4 Alerting & Reporting Architecture

- **Budget Alerting:** A single aws_cloudwatch_metric_alarm is configured in the us-east-1 region. It monitors the EstimatedCharges metric for the entire account. When the defined threshold is breached, it publishes a message to an SNS topic. The SNS topic has an email subscription, which then notifies the designated recipient.
- **Scheduled Reporting:** A aws_cloudwatch_event_rule is configured with a cron expression (cron(0 9 ? * MON *)). This rule triggers the GetWeeklyCostReport Lambda function every Monday at 9:00 AM UTC.

- **Report Generation:** This Lambda function performs a similar Cost Explorer query, formats the results into a detailed HTML email, and uses Amazon SES to send the report.
- **Resilience:** Both Lambda functions are configured with a Dead Letter Queue (DLQ) using Amazon SQS. If a function invocation fails after the default retries, the failed event payload is sent to its respective SQS queue for offline debugging.

3.0 Infrastructure as Code (Terraform)

The entire infrastructure is managed in a single Terraform environment located in the dev/ directory.

3.1 Directory Structure & Modularity

The configuration is heavily modularized to promote reusability and separation of concerns.

- dev/main.tf: The root module, responsible for orchestrating the creation of resources by calling child modules. It handles the "wiring" between modules (e.g., passing a KMS key ARN from the kms module to the lambda module).
- dev/modules/: Contains reusable modules for each logical component (API Gateway, CloudFront, IAM, KMS, Lambda, S3, SES, SNS). This approach ensures that a change to a component (e.g., adding encryption to S3) is made in only one place.
- **Dependency Management:** Dependencies are managed both implicitly (by passing outputs as inputs) and explicitly (depends_on) where necessary to resolve complex dependencies, such as the S3/CloudFront cycle.

3.2 Key Design Decisions

- **Randomized Suffixes:** The random_id provider is used to append a unique suffix to globally unique resources (like S3 buckets) to prevent naming collisions and enable multiple deployments of the same code.
- **Dynamic Data Sources:** aws_caller_identity and aws_region data sources are used within modules to dynamically construct ARNs and service principals without hardcoding account-specific information.
- **Conditional Creation:** The reusable Lambda module uses count meta-arguments (count = var.schedule_expression != null ? 1 : 0) to conditionally create CloudWatch

Event resources, allowing the same module to be used for both scheduled and API-triggered functions.

- **Security Suppressions:** Non-applicable Checkov findings are formally suppressed using in-line #checkov:skip=... comments with clear justifications, acknowledging the findings and documenting the accepted risk.

4.0 CI/CD Pipeline (GitHub Actions)

The project utilizes two GitHub Actions workflows for complete lifecycle automation.

4.1 terraform-deploy.yml (Deployment Workflow)

- **Trigger:** Activates on a push to the main branch or a pull_request targeting main.
- **Authentication:** Uses OpenID Connect (OIDC) to securely authenticate with AWS. This is a best practice that avoids storing long-lived IAM access keys as GitHub secrets.
- **Jobs & Flow:**
    1. **terraform Job:**
        - Checks out the source code.
        - Runs static analysis tools (tflint for best practices, checkov for security). A failure in tflint will stop the pipeline. checkov is set to soft_fail to allow review of non-critical findings.
        - Runs terraform init and terraform validate.
        - **If a Pull Request**, it runs terraform plan and posts the output as a PR comment for peer review.
        - **If a Push to main**, it runs terraform apply -auto-approve to provision the infrastructure.
        - Finally, it extracts key outputs (s3_bucket_name, api_endpoint_url) and exposes them to the workflow.
    2. **deploy-frontend Job:**
        - This job needs: terraform and only runs if the terraform job is successful on a push to main.
        - It receives the S3 bucket name and API endpoint URL from the terraform job's outputs.
        - It uses sed to perform a **runtime injection**, replacing a %%API_ENDPOINT%% placeholder in script.js with the live API Gateway URL.

- It uses aws s3 sync to upload the frontend assets to the S3 bucket, including Cache-Control headers to prevent browser caching issues.

4.2 terraform-destroy.yml (Destroy Workflow)

- **Trigger:** Manual (workflow_dispatch) only, for safety.
- **Safety Gate:** Requires the user to type the case-sensitive word "destroy" into an input field to confirm intent. An explicit verification step checks this input before proceeding.
- **Action:** Runs terraform destroy -auto-approve to cleanly tear down all infrastructure managed by the dev workspace.

5.0 Operational Procedures & Maintenance

5.1 Making an Infrastructure Change

1. Create a new feature branch from main.
2. Make the desired Terraform code changes in the dev/ directory.
3. Push the branch and open a Pull Request.
4. The pipeline will automatically run the static analysis tools and post a terraform plan to the PR.
5. Review the plan. Once approved, merge the PR.
6. The pipeline will then automatically run on the main branch and apply the changes.

5.2 Making a Frontend Change

1. Follow the same branch/PR process, making changes to the files in frontend/public/.
2. The terraform job will run but will show "No changes" in the plan.
3. Once merged, the deploy-frontend job will automatically sync the new frontend files to the S3 bucket.

5.3 Troubleshooting

- **Frontend Errors (Failed to Load):**
    1. Verify the API Gateway integration in the AWS Console to ensure it's pointing to the correct Lambda.

2. Check the CloudWatch Logs for the GetCostDataApi Lambda function for any execution errors.
3. Check the API Gateway access logs for 4xx or 5xx status codes.

- **Missing Email Reports:**
  1. Check the CloudWatch Logs for the GetWeeklyCostReport Lambda function.
  2. Check the SQS Dead Letter Queue associated with the function for any failed invocation messages.
  3. Ensure the email identity is still "Verified" in the SES console.