# FinOps Case Study: Static Website on AWS

## Overview

This FinOps case study analyzes the total cost of ownership and optimization opportunities for a static website architecture hosted entirely on AWS. The infrastructure serves a small to medium-scale workload with modest monthly visitor traffic.

## Infrastructure Summary

| Layer | AWS Resources Used |
| --- | --- |
| **Frontend** | Static HTML/CSS/JS hosted on Amazon S3 |
| **CI/CD** | GitHub Actions with dry-run testing, fallback logic |
| **Storage** | S3 with bucket policies (no ACLs) |
| **Networking** | CloudFront CDN, Route 53 DNS |
| **Serverless** | Lambda Function URL for visitor tracking |
| **Database** | DynamoDB (on-demand, TTL enabled) |
| **Monitoring** | S3 access logs, CloudWatch for Lambda |
| **Security** | IAM roles, least privilege policies, CORS headers |

## Traffic Assumption

* **Baseline Load**: 2,500 visitors/month
* **Payload**: ~1MB per visitor
* **Lambda Execution**: 120ms x86, 128MB
* **Flat Costs**: Route 53, IAM roles

## Monthly Cost Estimation @ 2,500 Visitors

This chart breaks down AWS service costs before and after implementing cost optimization measures. Key actions include:

* Enabling S3 intelligent tiering
* Optimizing CloudFront cache policies
* Reducing DynamoDB read/write units
* Streamlining Lambda memory and execution time

## Annualized Cost Projection

The yearly projection demonstrates how small savings compound:

* Over $50/year saved on CloudFront
* Over $40/year saved on Lambda
* Nearly 20% overall reduction in total infrastructure cost

## Additional Visualizations

### 🔁 Cost Allocation Flow (Sankey)

This diagram shows how each AWS service maps to a functional layer. Helps with stakeholder education and chargeback.

### 📉 Cost Optimization Impact (High-Traffic Scenario)

Based on 50,000 monthly visitors, this chart illustrates the maximum impact of FinOps practices under higher load.

## Summary & Insights

* AWS cost is predictable and scalable for static workloads.
* Optimization at even small scale (2,500 visitors/month) yields meaningful annual savings.
* Leveraging AWS Budgets and dashboards provides visibility and control.
* Flat services (like Route 53) are predictable, while variable services (like CloudFront, Lambda) benefit most from tuning.

## Next Steps

* Integrate this report into stakeholder dashboards
* Monitor real-time usage via CloudWatch
* Continue tracking cost anomalies with AWS Budgets
* Evaluate Reserved Capacity if traffic grows

Prepared by: *[Your Name or Org]*  
Date: July 2025

## Stack by Stack Breakdown

* **Serverless backend**
  + ✅ **FinOps**:
    - No idle capacity cost via PAY\_PER\_REQUEST
    - TTL reduces long-term storage
    - Lightweight x86 runtime for cheaper compute
* **Static frontend**
  + ✅ **FinOps**:
    - S3 + CloudFront = ultra low cost delivery
    - No managed DNS = reduced monthly bill
* **Dedicated log storage bucket**
  + Lifecycle rule: logs expire after 60 days
  + Bucket policy: allows S3 logging from AWS
  + Parameterized via LogBucketName
  + ✅ **FinOps**:
    - Prevents long-term log storage charges
    - Centralizes and simplifies access logging
* **Basic Observability**
  + **CloudWatch Alarm** on Lambda errors
  + No paid SNS or email integrations — alerts sent via GitHub Actions/Slack
  + AWS Budgets are already configured
  + ✅ **FinOps**:
    - No AWS-native alerting cost (Slack + GitHub notify)
    - Keeps observability minimal but effective

## ✅ FinOps Highlights

Embedded into every template and decision:

* **Minimal runtime execution** (Lambda, DynamoDB, TTL)
* **Pay-as-you-go services** only
* **No premium AWS integrations** like Chatbot, SES, SNS
* **Lifecycle expiration policies** on log storage
* **Free alerting via GitHub & Slack integrations**