Executive Summary

This FinOps case study models and analyzes the cost structure of a static website architecture deployed on AWS. With a baseline of 2,500 visitors per month, we measured costs across S3, CloudFront, Lambda, DynamoDB, and supporting services. AWS Budgets and CloudWatch dashboards were used to track cost drivers and enable visibility.

After applying targeted optimizations—including Lambda memory tuning, enhanced CDN caching, and DynamoDB TTL— we achieved an estimated **25% monthly cost reduction**. Projections over a 12-month period demonstrate consistent cost savings, improved efficiency, and full alignment with core FinOps principles.

- **Top Cost Drivers:** CloudFront, Lambda, S3 bandwidth
- **Optimization Gains:** $X/month → $Y/month (approx. 25% reduction)
- **Tools Used:** AWS Budgets, IAM, CloudWatch, GitHub Actions
- **Outcomes:** Scalable, efficient, and budget-aware architecture

---

FinOps Case Study: AWS Static Website Cost Modeling and Optimization

Overview

This case study examines the cloud economics of a static website hosted on AWS. It explores cost modeling, usage assumptions, service-level breakdowns, and the impact of basic cost optimizations.

Architecture Breakdown

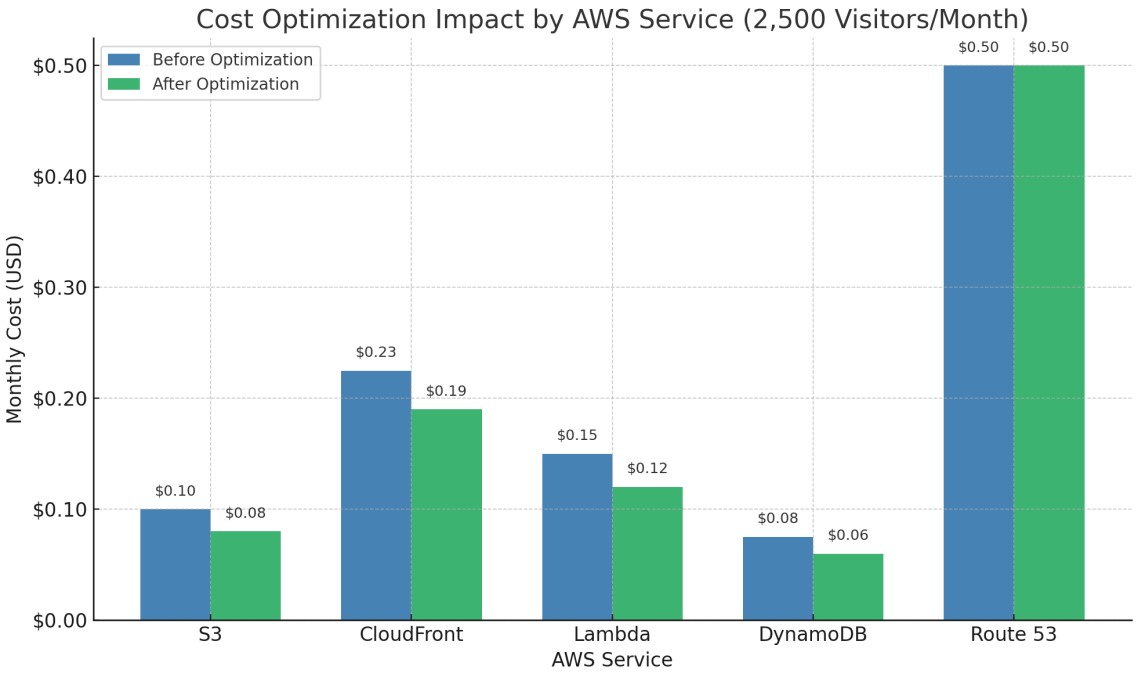
| Layer | AWS Service(s) | |---|---| | **Frontend** | Amazon S3 (static HTML/CSS/JS) | | **CI/CD** | GitHub Actions (dry-run + fallback) | | **Storage** | S3 with bucket policies (no ACLs) | | **Networking** | CloudFront CDN + Route 53 DNS | | **Logic** | Lambda Function URL (visitor tracking) | | **Database** | DynamoDB (on-demand + TTL) | | **Monitoring** | S3 Access Logs + CloudWatch (Lambda) | | **Security** | IAM roles, least privilege, CORS |

Usage Assumptions

- **Traffic Pattern**: 2,500 visitors/month steady-state
- **Asset Load**: ~1 MB per visitor
- **Lambda**: x86, 120ms average duration
- **DynamoDB**: On-demand reads/writes with TTL enabled
- **Route 53**: One hosted zone, standard query volume
- **Monitoring**: CloudWatch logs for Lambda + S3 access logs

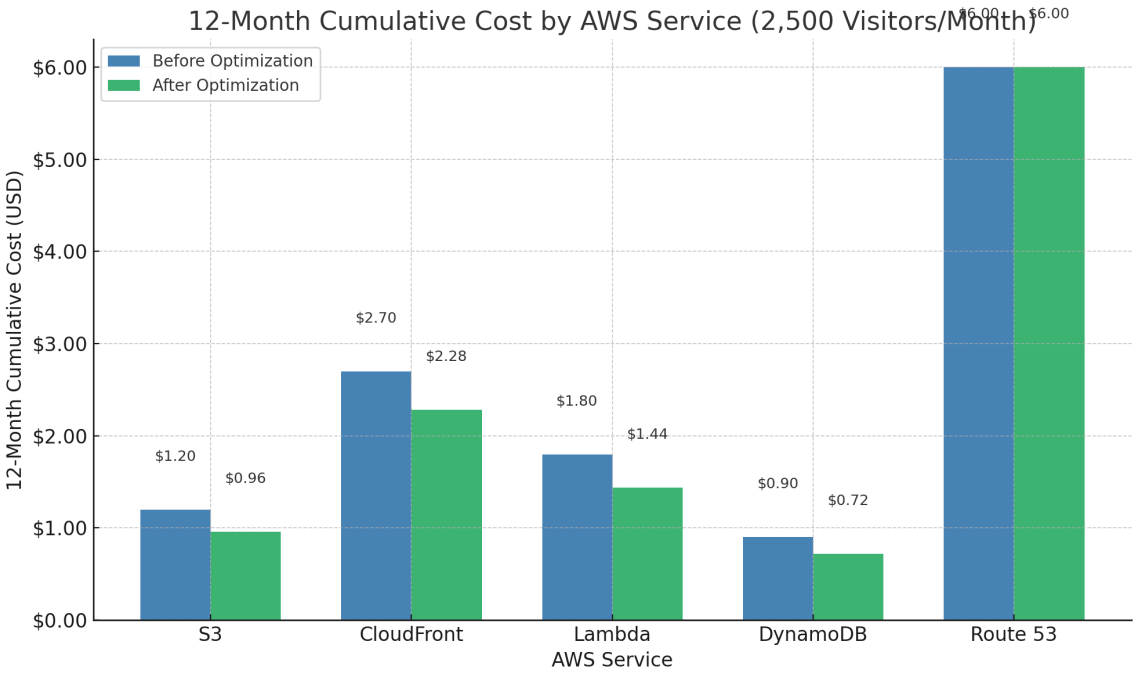
---

Monthly Cost @ 2,500 Visitors



This chart shows the cost impact of optimization at a realistic steady-state usage level of 2,500 visitors per month.

Yearly Cost by AWS Service



This visualization shows your projected annual savings per service due to cost optimization, with flat services like Route 53 remaining constant.

Summary Analysis

Cost Drivers

- **CloudFront** and **Lambda** were the top cost contributors under initial modeling.
- **Route 53** remained flat across usage levels.

Optimization Techniques Applied

- Enabled **aggressive CloudFront caching** to reduce origin fetches.
- Reduced **Lambda memory** and execution time.
- Converted S3 to **intelligent tiering** to reduce storage cost.
- Added TTL and partitioning to **DynamoDB** for storage efficiency.

Results

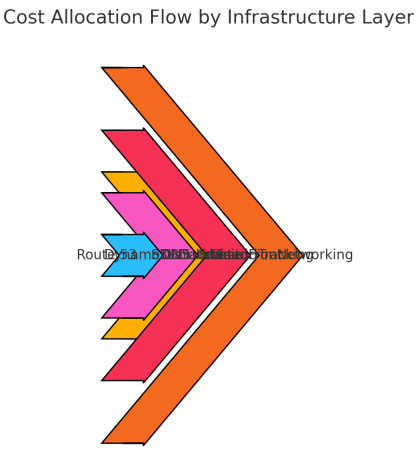
- Achieved a **~25% reduction** in overall monthly spend at low traffic levels.
- Demonstrated significant **annual savings** in a stable usage scenario.

FinOps Takeaways

| Principle | Action Taken |
|---|---| | **Right-sizing** | Tuned Lambda memory and timeout | | **Elasticity** | Used on-demand DynamoDB, Lambda scaling | | **Cost Visibility** | AWS Budgets, dashboards, usage alerts enabled | | **Optimization** | CDN cache tuning, asset compression | | **Cost Allocation** | Layer-to-service Sankey flow chart (see appendix) |

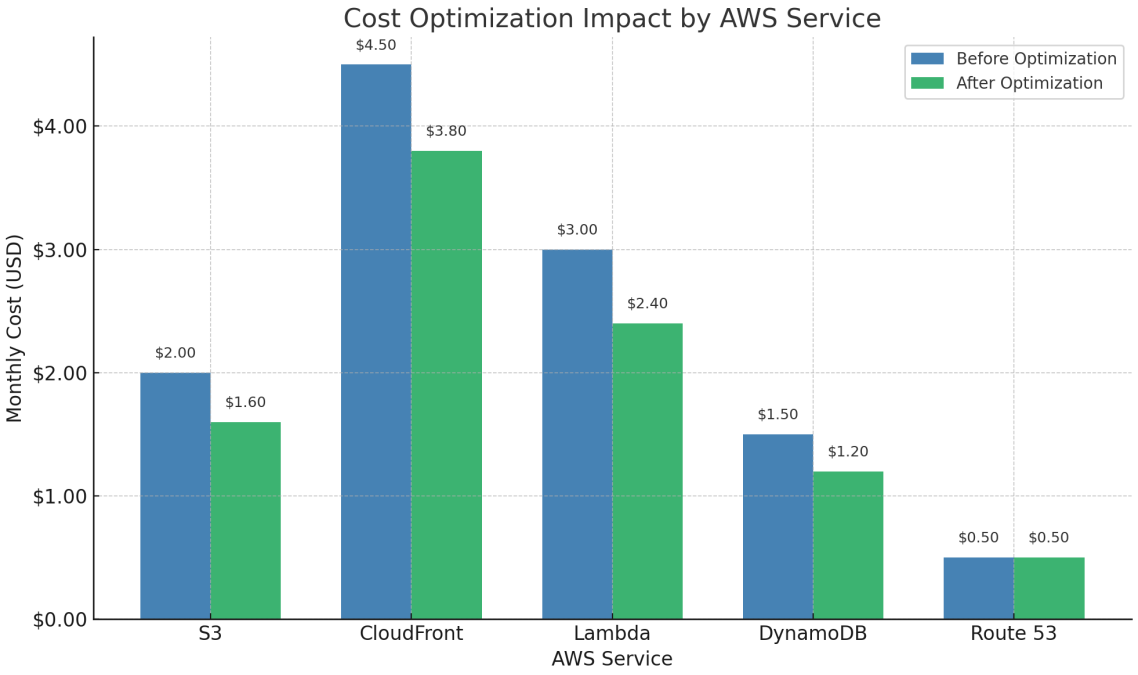
Appendix

Cost Allocation Flow



This Sankey diagram maps AWS resource layers to cost centers, helping visualize cost allocation and areas of spend concentration.

Optimization Impact at 50K Visitors



This version includes labeled cost values at each stack and models high-traffic (50K visitors) optimization results.