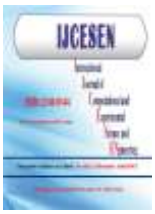
International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 12-No.1 (2026) pp. 786-798
http://www.ijcesen.com



ISSN: 2149-9144

Research Article

Engineering-Driven Cloud Cost Optimization at Enterprise Scale: An Applied Success Story with Measured Outcomes in a Large Healthcare Enterprise

Jayasree Natarajan Swarnaras*

Independent Researcher, USA
* **Corresponding Author Email:** connectwithjayasree@gmail.com - **ORCID:** 0000-0002-5247-7660

Article Info:

DOI: 10.22399/ijcesen.4917
Received : 15 December 2025
Revised : 25 January 2026
Accepted : 02 February 2026

Keywords

Cloud Cost Optimization;
Finops,
Site Reliability Engineering,
Resource Rightsizing,
Autoscaling,
Healthcare IT

Abstract:

Enterprises continue to experience rising cloud infrastructure costs as application portfolios expand and cloud-native architectures proliferate. This paper presents a production-validated, engineering-driven cloud cost optimization framework implemented at a large U.S. healthcare enterprise operating a multi-account, multi-region platform with unpredictable demand, strict compliance requirements, and high-availability expectations. The framework integrates utilization-based rightsizing, demand-aware autoscaling, storage lifecycle management, commitment-based pricing, and continuous governance through policy-as-code, explicit cost ownership, and cost–performance observability, embedding cost optimization into routine operations via automated enforcement and recurring review checkpoints. Over two fiscal years, the initiative reduced annualized infrastructure cost by approximately 18% in year one and an additional 5% in year two, despite continued growth in platform demand and overall spend, while availability, latency, and error rates remained within established service-level objectives. The results demonstrate that cloud cost efficiency can be operationalized as a continuous engineering discipline – complementing site reliability engineering practices – rather than treated as an episodic financial exercise, and provide a repeatable, scalable model for enterprises seeking measurable and sustainable optimization.

1. Introduction

Public cloud adoption has accelerated across industries as organizations pursue scalable, resilient, and rapidly deployable application platforms. However, the proliferation of cloud-native architectures has introduced significant complexity in managing operational costs, often undermining the economic benefits originally anticipated. Industry analyses, including the Bessemer Venture Partners State of the Cloud report, indicate that efficiency has become a critical concern for cloud-native organizations, with enterprises shifting focus from growth at scale to sustainable, cost-efficient operations [1].

Existing academic and practitioner literature largely emphasizes theoretical optimization models or isolated tooling approaches, offering limited insight into how cost optimization strategies can be sustained at enterprise scale in production environments. Prior work examining the economics of cloud and serverless computing highlights that while cloud platforms provide flexibility and elasticity, certain workload patterns and architectural choices can result in unexpectedly high costs compared to traditional infrastructure models [2]. These findings underscore the need for cost optimization to be addressed as a systemic, engineering-driven problem rather than a collection of ad hoc interventions.

In regulated domains such as healthcare, cost optimization is further constrained by strict compliance requirements, high-availability expectations, and patient-critical service-level objectives (SLOs). This paper presents a production-tested, end-to-end cloud cost optimization framework implemented within a multi-account, mission-critical healthcare platform characterized by variable demand patterns. The framework integrates workload-aware scaling, infrastructure rightsizing, storage lifecycle policies, commitment-based pricing, and continuous

governance mechanisms, including policy-as-code, explicit cost ownership, and cost–performance observability.

This paper makes three primary contributions:

1. An engineering-driven optimization framework that combines technical controls with organizational governance.
2. Measured, multi-year financial and operational outcomes demonstrating sustained cost reduction without performance degradation.
3. A repeatable model to prevent post-optimization regression by embedding cost accountability and enforcement into standard operational workflows.

2. Problem Context and Research Motivation

The subject environment is a distributed, multi-account, multi-account public cloud platform supporting mission critical healthcare applications. Demand exhibits pronounced diurnal, seasonal, and event driven variability, with unpredictable spikes. Rapid service growth, decentralized ownership across numerous application teams, and uneven cost accountability led to sustained infrastructure cost escalation that threatened the economic rationale for cloud adoption. The platform comprises thousands of compute instances, dozens of managed databases, and petabyte scale storage distributed across geographic regions, creating a broad and interdependent optimization space that requires coordinated intervention.

Prior healthcare cloud studies focus primarily on security and compliance, with limited longitudinal analysis of cost efficiency outcomes. Conventional cost cutting measures proved insufficient at scale. Manual rightsizing and one-off cleanup campaigns generated temporary relief but introduced operational risk, required significant human effort, and frequently regressed as teams evolved. Budget caps and spending alerts provided limited guidance: they obscured the distinction between mission critical and non-critical consumption and did not prevent inefficient resource patterns (e.g., over provisioned compute to peak, idle premium storage, unaligned commitments). Industry surveys consistently report that optimizing existing cloud use and managing cloud spend remain top priorities and persistent challenges for enterprises, highlighting the need for systemic solutions rather than tactical fixes [3]. These observations indicate that cost governance, not just isolated technical remediation, is the dominant failure mode.

The healthcare context compounds these challenges. Compliance obligations and the protection of sensitive data impose strict controls, while high availability (HA) expectations and patient critical SLOs constrain the aggressiveness of optimization strategies. Prior analyses of cloud economics note that certain workload patterns and architectural decisions can yield unexpectedly high costs relative to traditional infrastructure, despite cloud elasticity [2], and healthcare specific studies emphasize that security, legal, and availability requirements limit optimization flexibility while elevating baseline costs [4]. Consequently, there is a clear need for a holistic, engineering driven framework that integrates technical levers with explicit governance, including policy as code, cost ownership, and cost–performance observability to achieve repeatable and sustained outcomes without compromising reliability.

3. Related Work and Background

Prior work and industry practice have explored individual optimization mechanisms, including utilization-based rightsizing, autoscaling, storage lifecycle tiering, and commitment-based pricing, often demonstrating benefits in constrained settings. However, the literature tends to emphasize theoretical models or tool-specific techniques with limited evidence of long-term durability in production at enterprise scale [1], [2].

Existing FinOps and cloud cost optimization research can be categorized into three primary streams, each with distinct limitations that this work addresses:

3.1 Technical Optimization Studies:

Iqbal et al. [4] and Roy et al. [5] present adaptive resource provisioning and predictive autoscaling models demonstrating efficiency gains in controlled environments, while AWS and cloud provider documentation [2] offers best-practice guidance on rightsizing and commitment strategies. These contributions focus predominantly on individual optimization techniques, autoscaling, rightsizing, or commitment planning—evaluated in isolation or simulation. In contrast, this work implements a coordinated multi-pillar framework in a production healthcare environment, demonstrating how simultaneous application of complementary techniques yields compounding benefits and prevents cost leakage across the optimization surface. Moreover, prior studies report initial efficiency improvements but provide limited evidence of sustained outcomes beyond 6–12 months; this work presents two-year longitudinal results with demonstrated regression prevention.

3.2 FinOps Governance and Organizational Practice:

Industry reports from McKinsey [6], the FinOps Foundation, and cloud maturity frameworks emphasize governance mechanisms such as showback/chargeback, tagging taxonomies, and cross-functional collaboration to align engineering and finance [1], [3], [7]. While these frameworks establish valuable organizational principles, they typically lack detailed technical implementation specifications or empirical validation in production systems. Published case studies often report qualitative adoption challenges or cultural shifts without quantifying sustained financial outcomes or operational impact. This work bridges that gap by presenting an operationalized governance architecture—policy-as-code enforcement, mandatory cost ownership via CI/CD gates, and cost–performance observability—integrated with technical controls, and validates the approach with measured multi-year cost reductions and maintained SLOs.

3.3 SRE and Reliability-Cost Integration:

The Site Reliability Engineering literature and Google's SRE practices emphasize error budgets, observability, and resilience engineering but treat cost efficiency as secondary to availability and latency objectives [8]. Recent DevOps and SRE surveys acknowledge growing interest in cost as a reliability dimension [7], yet few studies operationalize cost optimization as a first-class SRE discipline with the same rigor applied to availability or performance. This work explicitly positions cost efficiency as complementary to SRE practices, embedding cost observability, automated guardrails, and blameless post-optimization reviews into standard reliability workflows, demonstrating that cost and reliability objectives can be jointly optimized rather than traded off.

3.4 Healthcare Cloud Studies:

Prior healthcare-focused cloud research emphasizes security, compliance, and clinical data protection [3], with limited analysis of cost efficiency outcomes in production environments subject to HIPAA, HA, and patient-critical SLOs. Existing studies acknowledge that regulatory constraints elevate baseline costs and narrow optimization flexibility but provide minimal guidance on achieving measurable cost reductions within these constraints. This work directly addresses regulated healthcare environments by presenting production-validated techniques that respect compliance boundaries while achieving double-digit cost reductions.

3.5 Positioning and Novelty:

While individual elements of the proposed framework—rightsizing, autoscaling, lifecycle policies, commitments, and governance practices—exist in prior literature, empirical demonstrations that (i) integrate these techniques into a unified, policy-enforced system, (ii) implement the framework in a mission-critical regulated environment, (iii) preserve reliability SLOs throughout optimization, (iv) report sustained multi-year cost reductions despite platform growth, and (v) prevent regression through automated governance remain comparatively rare in the public domain. This work addresses that gap by operationalizing cost efficiency as a continuous engineering discipline aligned with SRE practices, combining technical controls with policy-as-code governance, and presenting longitudinal financial and operational outcomes in a large healthcare enterprise that simultaneously validate feasibility, durability, and repeatability.

4. System and Environment
Platform Scope

The evaluated platform supports a portfolio of patient-facing and clinical applications with diurnal, seasonal, and event-driven load variability. The estate spans multiple regions and organizational accounts and includes thousands of compute resources (VMs and containerized workloads), dozens of managed database services, and petabyte-scale object and block storage. Services are owned by independent application teams, with a central platform/SRE group providing shared infrastructure, guardrails, and governance.

4.1 Baseline State and Anti-Patterns

Prior to the initiative, several cost-inefficient patterns were prevalent: (1) static over-provisioning to peak demand, (2) limited rightsizing discipline and drift over time, (3) dormant or low-temperature data retained on premium storage tiers, (4) under- or mis-aligned commitments relative to workload baselines, and (5) unclear cost ownership and inconsistent tagging, which reduced the effectiveness of spend analysis and accountability. These conditions reflected a combination of technical debt and organizational fragmentation.

4.2 Constraints

Optimization levers were bounded by regulatory and compliance requirements, HA expectations, and patient-critical SLOs. Changes to capacity, deployment topology, or storage policy required controlled experimentation, progressive rollout, and observability to ensure no degradation in availability, latency, or error rates.

4.3 Operational Context and Observability Infrastructure

The platform maintained unified observability and cost telemetry infrastructure comprising multiple integrated layers. Metrics collection leveraged a Prometheus-compatible time-series database with custom exporters capturing CPU, memory, network I/O, disk utilization, and application-layer golden signals (request rate, error rate, latency distributions) at 15-second granularity across all compute resources. Distributed tracing via OpenTelemetry instrumentation enabled end-to-end request flow analysis and service dependency mapping. Log aggregation centralized application and infrastructure logs with structured metadata for filtering and correlation. Cost telemetry integrated native cloud provider billing APIs and cost allocation tags, correlating resource consumption with financial spend at hourly granularity and enabling drill-down to individual service, team, environment, and workload dimensions.

This observability substrate was augmented by a unified analytics layer that joined cost, utilization, and performance data across services and time windows. Custom dashboards and alerting rules enabled real-time cost–performance correlation, allowing engineers to identify optimization opportunities (e.g., sustained low CPU utilization indicating rightsizing candidates) and detect anomalies (e.g., cost spikes decoupled from traffic growth). Historical data retention of 18+ months supported trend analysis, seasonality detection, and validation of demand forecasting models used in predictive autoscaling.

A central policy-as-code layer enforced guardrails including instance class allowlists, mandatory autoscaling configurations for production workloads, storage lifecycle defaults, and commitment hygiene rules, with automated drift detection comparing deployed state against desired policy. Recurring cost reviews and automated validation checks were embedded into standard operational workflows, including weekly team-level spend reviews, monthly cross-team optimization forums, and quarterly architecture governance sessions requiring documented cost impact assessments for significant infrastructure changes. This environment provided the necessary substrate—observability, governance tooling, and operational discipline—to implement and validate a continuous cost optimization program without jeopardizing reliability.

5. Optimization Framework and Methodology

The proposed framework operationalizes cloud cost optimization as a continuous, engineering-driven process rather than a one-time corrective exercise. Optimization actions are derived from telemetry-guided analysis, validated against service-level objectives (SLOs) through controlled experimentation, and enforced via automated pipelines and policy-as-code governance to ensure durability. The framework integrates six complementary technical pillars spanning the infrastructure, platform, and governance layers.

5.1 Utilization-Based Rightsizing

Utilization telemetry across CPU, memory, network, and storage, collected over extended historical windows (30–90 days) and supplemented by near-real-time signals, served as the basis for rightsizing recommendations. Analysis workflows combined Prometheus query results with custom Python-based analytics scripts that identified resources exhibiting sustained underutilization—defined as CPU utilization consistently below 40% or memory utilization below 50% during business-hours peaks, with no significant variability across daily or weekly cycles.

Resources meeting underutilization criteria were mapped to more appropriate instance classes or sizes using a decision matrix that considered workload characteristics (CPU-bound, memory-bound, network-intensive, or balanced) alongside cost-performance ratios across available instance families. Representative rightsizing actions included:

API gateway tier: Transitioned from m5.2xlarge (8 vCPU, 32 GiB) to m5.xlarge (4 vCPU, 16 GiB) instances after observing sustained CPU utilization of 25–30% and memory at 35–40%, achieving 42% unit cost reduction while maintaining p95 latency < 100ms and 99.95% availability.

Background processing workloads: Migrated from general-purpose m5.large to compute-optimized c5.large instances for CPU-intensive batch jobs, reducing per-instance cost by 8% while improving job completion time by 12% due to higher CPU performance.

Database read replicas: Downsized from r5.4xlarge to r5.2xlarge (memory-optimized) after sustained memory utilization remained below 45%, yielding 50% cost savings with query latency degradation < 5ms (within acceptable thresholds).

Recommendations were applied through canary deployments: changes were first validated on 5% of fleet capacity, monitored for 48–72 hours against predefined golden KPIs (availability ≥ 99.9%, p95 latency within ±10% of baseline, error rate < 0.1%), and progressively rolled out to 25%, 50%, and 100% of capacity upon successful validation at each stage. Automated rollback triggered if any KPI threshold was breached. This deterministic, workload-aware provisioning replaced static allocation models that historically led to chronic over-provisioning in variable workloads. Prior studies have demonstrated that utilization-driven adjustments based on workload characteristics significantly outperform static provisioning strategies in both efficiency and stability [5].

5.2 Demand-Aware Autoscaling

To address inefficiencies introduced by fixed capacity planning, the framework implemented demand-aware autoscaling combining predictive, scheduled, and reactive mechanisms. Historical demand patterns (request volume, queue depth, database connection counts) were analyzed using time-series forecasting models (ARIMA and exponential smoothing) trained on 90-day rolling windows to inform predictive scaling models, enabling capacity adjustments 15–30 minutes ahead of anticipated load changes.

Specific autoscaling configurations implemented included:

Scheduled scaling for patient portal applications: Automatic scale-up from baseline 20 instances to 60 instances at 06:00 local time (ahead of peak clinic appointment check-in hours), scale-down to 25 instances at 20:00, and further reduction to 15 instances overnight and weekends, eliminating 40% of off-hours capacity waste while maintaining sub-second response times during traffic surges.

Predictive scaling for data processing pipelines: Machine learning model predictions triggered proactive scaling 20 minutes before batch job submissions (detected via queue depth monitoring), reducing scale-up latency from 8–10 minutes (reactive threshold-based) to < 2 minutes and eliminating 15% of job failures caused by insufficient capacity during demand spikes.

Reactive autoscaling with bounded limits: Horizontal pod autoscalers (HPA) in Kubernetes clusters configured with target CPU utilization of 70%, minimum 3 replicas, maximum 50 replicas, and 2-minute stabilization windows to prevent oscillation. Scale-up velocity limited to +50% per 5-minute interval; scale-down limited to −25% per 10-minute interval to ensure safe warm-up and graceful shutdown.

Where supported, vertical scaling (vertical pod autoscaler in Kubernetes) complemented horizontal instance scaling for stateful workloads, adjusting CPU and memory requests/limits within bounds of 0.5–8 vCPU and 2–32 GiB based on observed utilization, bounded by safety limits and warm-up controls to prevent destabilizing memory-intensive processes. Research indicates that proactive scaling approaches improve cost efficiency and performance consistency compared to purely reactive threshold-based methods by reducing scale-up latency and capacity shortfalls during demand surges [6].

5.3 Storage Lifecycle Optimization

Storage optimization targeted the disproportionate cost impact of retaining low-activity data on premium tiers. Access-frequency analysis queried cloud provider storage analytics APIs to identify objects with zero access in trailing 30-day, 90-day, and 180-day windows. Retention requirements and recovery objectives (RTO/RPO) informed automated lifecycle policies that transitioned data across hot (standard SSD), warm (infrequent access), cold (archival with hours retrieval), and archival (deep archive with 12-hour retrieval) tiers.

Lifecycle transition rules included:

Medical imaging archives: Transition from hot to warm storage after 90 days of zero access, warm to cold after 180 days, and cold to deep archive after 365 days, subject to 7-year legal retention and 24-hour RTO compliance requirements, achieving 65% storage cost reduction for datasets > 1 year old.

Application logs: Transition to infrequent access after 30 days, archive after 90 days, with automatic deletion after 2 years (retention policy), reducing log storage costs by 70% while maintaining audit compliance.

Highly accessed datasets remained on premium storage, while infrequently accessed or dormant data was progressively tiered down after defined inactivity periods. Exceptions were governed through a centralized registry where teams documented RTO/RPO requirements, compliance obligations, and business justifications for retaining data on higher-cost tiers, ensuring that lifecycle transitions reduced unit storage cost at scale without violating operational or regulatory requirements.

5.4 Commitment-Based Pricing Optimization

For workloads exhibiting stable baseline consumption, commitment-based pricing instruments (Reserved Instances, Savings Plans, Committed Use Discounts) were applied to capture provider discounts (20–40% off on-demand pricing) while minimizing over-commitment risk. Baseline utilization analysis examined 90-day minimum sustained capacity ("floor" consumption) across instance families and regions, excluding burst traffic and temporary capacity. Growth projections (historical trend analysis and business planning inputs) informed coverage decisions, targeting 70–80% of baseline capacity under commitments to retain flexibility for variable workloads and architectural changes.

Coverage decisions were reviewed monthly via automated reports comparing actual vs. committed utilization, triggering alerts when utilization fell below 85% (under-utilization risk) or exceeded 95% (opportunity for additional coverage). Commitments were adjusted quarterly in alignment with architectural changes (e.g., containerization reducing VM footprint) or sustained demand growth. This disciplined approach enabled sustained discount capture while retaining flexibility for variable workloads, achieving 28% effective cost reduction on committed capacity.

5.5 Idle Resource Remediation

Event-driven and scheduled detectors identified idle or orphaned resources using tag-based ownership validation and activity telemetry. Automated detectors queried cloud provider APIs nightly to identify:

- Unattached EBS volumes (storage not mounted to any instance) idle > 7 days
- Elastic IP addresses not associated with running instances > 3 days
- Load balancers with zero traffic > 14 days
- Database instances with zero connections > 7 days
- Snapshots exceeding retention policies

Automated remediation workflows enforced cleanup actions by default: resources flagged as idle received notifications to owning teams (via cost center tags) with 7-day grace periods, after which automated deletion occurred unless teams explicitly opted out via exception registry. Opt-out required documented business justification and executive approval for resources with monthly cost > $500. This capability addressed chronic accumulation of non-value-generating resources (representing 8–12% of baseline spend) and served as a guardrail against cost regression.

5.6. Governance and Orchestration

All optimization techniques were orchestrated through policy-as-code controls implemented using HashiCorp Sentinel (for Terraform infrastructure-as-code validation), AWS Config Rules (for runtime compliance), and Open Policy Agent (OPA) for Kubernetes admission control and integrated into continuous delivery pipelines. Policies enforced configuration standards including:

- Mandatory autoscaling for production workloads (HPA or ASG required)
- Storage lifecycle configuration for S3 buckets and persistent volumes
- Instance class allowlists (blocking oversized or previous-generation types)
- Commitment hygiene (minimum utilization thresholds, periodic review gates)
- Exception management workflows requiring documented approval chains

Automated validation checks in CI/CD pipelines blocked non-compliant infrastructure deployments, while runtime drift detection compared deployed state against desired policy every 15 minutes, triggering automated remediation or alerts for human review. Recurring cost reviews (weekly team-level, monthly cross-team forums, quarterly architecture governance) and automated validation checks embedded optimization into standard operational workflows, enabling cost efficiency to function as a continuous engineering discipline aligned with site reliability engineering (SRE) practices. Governance is treated as a first-class optimization pillar, providing enforcement, auditability, and regression prevention across all technical techniques. Detailed implementation of policy-as-code enforcement mechanisms, exception workflows, and organizational enablement is presented in Section 6.

6. Implementation Architecture and Governance

To ensure durability and prevent post-optimization regression, the cost optimization framework was operationalized through a governance-driven implementation architecture that embedded financial efficiency into standard engineering workflows. The architecture integrates explicit cost ownership, policy-as-code enforcement, cost–performance observability, operational checkpoints, and team enablement mechanisms, enabling cost efficiency to function as a continuous control loop rather than an episodic financial activity. Implementation required initial investment in tooling infrastructure, policy development, observability integration, and organizational

enablement, with setup effort distributed across platform engineering (6 engineer-months), policy authoring and testing (3 engineer-months), training development and delivery (2 engineer-months), and tooling licenses/integrations (estimated annual cost of $180K for policy engines, observability platform extensions, and cost analytics tools). These investments were amortized across the first fiscal year and offset by cost savings realized within the initial 4–5 months of operation.

6.1 Explicit Cost Ownership and Tagging

Cost accountability was decentralized to application teams, replacing centralized infrastructure ownership models. All provisioned resources were required to carry mandatory metadata including owning team, application, environment (dev/staging/prod), and cost center, enforced through CI/CD pipeline validation gates implemented as pre-deployment hooks in GitLab CI/CD and AWS CodePipeline. Tag validation logic queried resource definitions in Terraform plans and CloudFormation templates, blocking deployments missing required tags or containing non-compliant tag values (validated against organizational registries of approved teams, applications, and cost centers maintained in a central configuration management database).Non-compliant deployments were blocked at provisioning time with actionable error messages directing engineers to tagging documentation and team registration workflows, ensuring full attribution and enabling granular spend analysis across organizational, application, and environment dimensions. Tag compliance monitoring via scheduled AWS Config rules and custom scripts identified and flagged resources deployed outside CI/CD pipelines (manual console deployments), triggering automated notifications and requiring retroactive tag application within 48 hours or resource termination. This approach aligns with industry guidance emphasizing explicit accountability and cost awareness as prerequisites for sustainable cloud financial management [7].

6.2 Policy-as-Code Controls

Organizational cost optimization standards were encoded as executable policies and evaluated continuously at deployment and runtime using a multi-layer enforcement architecture: **Infrastructure-as-Code Validation (Pre-Deployment):** HashiCorp Sentinel policies integrated into Terraform Enterprise workflows evaluated infrastructure definitions before provisioning, enforcing rules such as:

- Instance type allowlists blocking previous-generation (t2, m4) or oversized instance classes (> 16 vCPU) without architectural review approval
- Mandatory autoscaling group or horizontal pod autoscaler configuration for production workloads (identified via environment tags)
- Required lifecycle policies on S3 buckets and persistent volume claims
- Commitment coverage targets requiring Reserved Instance or Savings Plan attribution for baseline-stable workload tiers

Runtime Compliance Monitoring: AWS Config Rules, Azure Policy, and custom Lambda functions evaluated deployed resource configurations every 15 minutes, detecting drift from desired state including:

- Autoscaling configurations disabled or improperly bounded (min/max instance count violations)
- Storage lifecycle policies removed or modified to retain data on premium tiers beyond approved durations
- Uncommitted compute instances exceeding 30-day runtime without approved exception
- Resource tag modifications or deletions violating ownership accountability

Kubernetes Admission Control: Open Policy Agent (OPA) deployed as a validating admission webhook intercepted pod creation requests, enforcing policies such as:

- Resource requests/limits within approved bounds (CPU: 0.1–8 vCPU, memory: 256 MiB–32 GiB)
- Required horizontal pod autoscaler definitions for production namespaces
- Pod disruption budgets ensuring safe scale-down operations
- Mandatory cost-allocation labels (team, application, environment)

Exception Management Workflow: Teams requiring policy deviations submitted exception requests via a self-service portal (ServiceNow integration), documenting business justification, expected duration, and estimated cost impact. Exception requests triggered an approval workflow routing to:

- Engineering manager approval for exceptions < $1K/month estimated impact and < 90 days duration
- Director-level approval for exceptions $1K–$10K/month or 90–180 days
- VP-level and architecture review board approval for exceptions > $10K/month or > 180 days

Approved exceptions were encoded as policy overrides (resource-specific exemptions or temporary policy disablement) with automated expiration, requiring renewal upon expiration or automatic policy re-enforcement. Exception telemetry (request volume, approval rates, cost impact, expiration compliance) was reviewed quarterly to identify systemic policy gaps or emerging architectural patterns requiring policy refinement. Automated drift detection identified deviations from desired states, triggering remediation or blocking unsafe changes. Consistent with prior industry findings, automated governance proved essential to sustaining cost efficiency without incurring ongoing manual oversight [8].

6.3 Cost–Performance Observability

Unified observability correlated cost telemetry with operational performance metrics, including availability, latency, throughput, and error rates, using integrated dashboards built on Grafana and custom analytics applications. Shared dashboards presented per-service and per-team views combining:

- Hourly infrastructure cost trends (compute, storage, data transfer) with 7-day and 30-day moving averages
- Key reliability metrics: availability (uptime percentage), p50/p95/p99 latency distributions, error rates, request volume
- Cost efficiency indicators: cost per request, cost per GB processed, cost per active user
- Optimization opportunity signals: underutilized resources flagged by rightsizing analyzers, autoscaling effectiveness scores, commitment coverage percentages

Engineers accessed real-time and historical views enabling cost-performance trade-off evaluation and quick identification of unintended regressions following optimization actions. Anomaly detection algorithms (statistical process control, seasonal decomposition) alerted on cost spikes decoupled from traffic growth or performance degradations coinciding with infrastructure changes, enabling rapid root cause analysis. When deviations were detected, teams conducted blameless post-optimization reviews following SRE incident review processes to refine policies and thresholds, document lessons learned, and adjust automation parameters, reinforcing reliability while maintaining cost controls.

6.4 Embedded Operational Checkpoints

Cost awareness was embedded into existing operational processes through standing checkpoints in team-level spend reviews and architecture governance forums:

Weekly Team-Level Spend Reviews: Engineering teams reviewed cost dashboards in weekly operational meetings, comparing actual spend against forecasts, investigating anomalies, and tracking progress on optimization initiatives. Reviews followed standardized agendas covering top cost-driving services, week-over-week variance analysis, and upcoming changes with cost impact.

Monthly Cross-Team Optimization Forums: Central platform/SRE group facilitated monthly forums presenting aggregate cost trends, sharing optimization patterns across teams, recognizing high-performing teams (cost efficiency awards), and coordinating platform-wide initiatives (e.g., commitment purchase negotiations, policy updates).

Quarterly Architecture Governance Reviews: Significant capacity or architectural changes (new service launches, major refactoring, region expansions) required documented cost impact assessments submitted to an architecture review board, including projected infrastructure spend, optimization strategies to be applied (rightsizing, autoscaling, lifecycle policies, commitment coverage), and comparison against alternative architectural approaches. Proposals lacking cost analysis or demonstrating inefficient resource patterns were returned for revision before approval.

Integrating cost reviews into established workflows ensured that optimization remained an ongoing engineering concern rather than a periodic remediation effort, making efficiency an explicit consideration alongside performance and reliability.

6.5 Team Enablement and Adoption

Successful operationalization of the cost optimization framework required coordinated team enablement addressing knowledge gaps, skill development, and behavioral incentives:

Training and Onboarding: A multi-tier training program was developed and delivered across the engineering organization:

Foundational training (2-hour self-paced modules): Cloud cost fundamentals, cost visibility tools, tagging requirements, and policy compliance expectations, required for all engineers provisioning infrastructure (completion tracked via LMS, 95% completion within 90 days of hire or role change)

Practitioner workshops (half-day instructor-led sessions): Hands-on exercises in rightsizing analysis, autoscaling configuration, storage

lifecycle policy authoring, and cost-performance trade-off evaluation using production datasets, targeting infrastructure and platform engineers (18 sessions delivered, 240 engineers trained in first year)

Advanced optimization training (full-day sessions): Deep dives into commitment-based pricing strategies, predictive scaling model development, policy-as-code authoring, and exception workflow design, targeting senior engineers and architects (quarterly offerings, 60 engineers trained in first year)

Training materials included recorded videos, interactive labs in sandboxed cloud environments, decision trees for optimization technique selection, and runbooks for common scenarios. Office hours (biweekly) provided ongoing support for teams encountering policy exceptions or optimization challenges.

Incentive Alignment: Cost efficiency was integrated into team and individual performance evaluation frameworks:

- Team-level cost efficiency targets (cost per request, cost per active user, or absolute spend budgets) incorporated into quarterly objectives and key results (OKRs), weighted at 15–20% of overall team goals
- Engineering managers included cost awareness and optimization contributions in individual performance reviews, recognizing engineers demonstrating sustained cost discipline or driving significant savings initiatives
- Quarterly "cost efficiency awards" recognized top-performing teams (greatest percentage reduction, most innovative optimization technique, best cross-team collaboration), providing visibility and informal recognition

Champions Network: A volunteer network of 25–30 "cost optimization champions" embedded across engineering teams served as local advocates, participated in policy refinement working groups, piloted new techniques before broad rollout, and provided peer mentoring. Champions received advanced training, monthly coordination meetings with the central platform team, and prioritized support for optimization experiments.

Cultural Reinforcement: Leadership communication (engineering all-hands presentations, internal blog posts, team newsletters) consistently reinforced cost efficiency as a core engineering value aligned with reliability and performance, celebrating wins and transparently sharing aggregate progress toward organizational cost targets. Blameless post-mortems for cost overruns (similar to incident reviews) normalized cost discussions and continuous improvement without punitive framing.

These enablement mechanisms reduced resistance to policy adoption, accelerated competency development, and sustained engagement with cost optimization as an ongoing engineering discipline rather than a transient compliance exercise.

6.6. Initial Investment and Setup Effort

Operationalizing the framework required upfront investment across multiple dimensions:

Tooling and Platform Infrastructure (estimated $180K annual recurring cost):

- Policy-as-code engine licensing (Terraform Enterprise with Sentinel: ~$50K)
- Observability platform extensions (Grafana Enterprise, custom analytics tools: ~$60K)
- Cost management and analytics platforms (cloud provider native tools augmented with third-party FinOps SaaS: ~$40K)
- CI/CD pipeline enhancements and integration development (~$30K in contractor support)

Engineering Effort (one-time setup, 11 engineer-months total):

- Platform engineering: Policy engine deployment, CI/CD integration, automation framework development (6 engineer-months)
- Policy development: Authoring, testing, and validating Sentinel/OPA/Config policies across use cases (3 engineer-months)
- Training development: Curriculum design, materials creation, lab environment setup (2 engineer-months)

Organizational Change Management:

- Executive sponsorship and communication (included in leadership time allocation, not separately costed)
- Training delivery (instructor time distributed across teams, estimated 0.5 engineer-months aggregate)
- Champion network coordination (volunteer time, minimal incremental cost)

These investments were amortized across the first fiscal year and offset by cost savings realized within 4–5 months of operation, with payback period significantly shorter than typical infrastructure projects. Ongoing operational overhead (policy maintenance, training updates, governance facilitation) required approximately 1.5 FTE sustained effort absorbed by the central platform/SRE team, representing < 2% of total engineering capacity while supporting enterprise-wide cost discipline.

7. Measurement and Evaluation Design

To assess the effectiveness and durability of the proposed optimization framework, a structured measurement and evaluation methodology was applied, emphasizing comparability, attribution accuracy, and protection of operational integrity.

7.1 Baselines and Evaluation Windows

Pre-implementation baselines were established using multi-month rolling averages to smooth seasonal and short-term variability. Post-implementation evaluation windows were aligned with fiscal periods to reflect business-relevant outcomes.

To isolate efficiency gains from organic platform growth, results were normalized using workload demand proxies (e.g., request volume, data processed, and storage footprint). Proxy selection and validation methodology: Candidate demand indicators were evaluated for correlation strength with infrastructure cost using 18-month historical data. Pearson correlation analysis identified request volume ($r = 0.87$), data processed ($r = 0.91$), and storage footprint ($r = 0.84$) as high-correlation proxies. A composite demand index was constructed as a weighted average of these three proxies (45%, 35%, 20% respectively, weighted by cost driver distribution), achieving overall correlation $r = 0.93$ with total infrastructure cost. The composite index was validated against a 3-month holdout period via regression analysis ($R^2 = 0.86$), confirming it explained 86% of cost variance and demonstrating robust normalization capability. This approach enabled comparison on a like-for-like basis despite increasing usage.

7.2 Attribution, Counterfactuals, and Scope

Cost savings were measured relative to a counterfactual run rate, adjusted for pricing commitments and growth trends that would have applied absent optimization. The evaluation scope was limited to infrastructure spend, excluding personnel, tooling, and licensing costs to avoid confounding factors and ensure attribution to technical and governance interventions.

7.3 Reliability and Safety Guardrails

System reliability was treated as a non-negotiable constraint throughout optimization. Availability, latency, and error rates were monitored continuously, with canary deployments and automated rollback mechanisms serving as safeguards. Any observed performance regression resulted in immediate rollback or pause of the associated optimization action for tuning, ensuring that cost improvements did not compromise service level objectives (SLOs).

8. Results

8.1. Financial Impact

In the first fiscal year following implementation, the program achieved an approximately 18% reduction in annualized infrastructure cost relative to the pre-implementation run-rate, while overall platform demand continued to grow. In the second year, on a larger demand and spend baseline, an additional ≈5% reduction was realized. This outcome indicates not only initial efficiency gains but also sustained prevention of cost regression in a maturing environment.

Qualitative attribution analysis showed that savings were distributed across multiple optimization categories, with utilization-based rightsizing contributing the largest share, followed by autoscaling efficiency, storage lifecycle optimization, commitment-based pricing, and remediation of idle resources. The distribution reinforces the value of a holistic, multi-pillar approach rather than reliance on a single optimization technique.

8.2 Operational Integrity

Throughout the optimization period, no material or sustained degradation was observed in application availability or latency relative to established baselines. No high-severity incidents were attributed to the optimization program. Incremental rollout, continuous monitoring, and automated rollback mechanisms collectively enabled cost efficiency improvements while preserving operational excellence.

8.3 Predictability and Forecast Accuracy

A secondary but operationally meaningful outcome was improved cost predictability. Post-implementation, variance between forecasted and actual infrastructure spend decreased materially, reducing budget volatility and friction between engineering and finance functions. Increased commitment coverage, standardized deployment patterns, and the elimination of runaway or idle resources contributed to more reliable annual planning and eliminated the need for mid-year budget corrections.

9. Discussion

An engineering-driven portfolio of techniques, reinforced by governance, produced material and durable cost outcomes without sacrificing operational excellence. Rightsizing and autoscaling addressed baseline waste; lifecycle policies corrected storage drift; commitment discipline monetized predictability; and policy-as-code deterred regression. The approach maps naturally to SRE practices: measure, change safely, verify, and enforce.

10. Threats to Validity and Limitations

Internal validity may be affected by baseline estimation and growth normalization choices; external validity is limited by the single-enterprise context. Construct validity is strengthened by triangulating financial outcomes with reliability and forecast metrics.

Scope limits exclude personnel and licensing costs. Reported cost savings represent infrastructure efficiency gains and do not account for the initial investment required to operationalize the framework, including tooling and platform infrastructure (~$180K annual recurring), engineering effort for setup and policy development (~11 engineer-months), and ongoing operational overhead (~1.5 FTE for policy maintenance and governance facilitation, detailed in Section 6). While these investments were offset by savings within 4–5 months, their exclusion from the primary savings calculations may overstate net financial benefit for organizations evaluating adoption without considering implementation costs. Public safety is maintained by reporting percentages and anonymizing the organization. Cloud provider-specific services and pricing models may affect the portability of individual techniques, though the governance and evaluation principles remain generalizable.

11. Practitioner Guidance

A. Implementation Checklist

- Enforce cost ownership via tags (team/app/env/cost-center) with CI/CD gates.
- Establish utilization baselines; generate and canary rightsizing recommendations.
- Combine predictive and scheduled autoscaling with safe bounds and warm-ups.
- Define storage tiering policies (hot/warm/cold/archive) with exceptions registry.
- Size commitment coverage to steady-state; review monthly to avoid over-commit.
- Encode standards as policy-as-code; enable continuous compliance and drift detection.
- Build cost–performance dashboards; alert on anomalous cost/perf drift.
- Add cost checkpoints to change and architecture reviews; track forecast variance.

Table 1 *summarizes the primary challenge categories, underlying issues, and their operational impact within the evaluated environment.*

Challenge Category	Key Issues	Impact
Infrastructure Complexity	Multi-account, multi-region; heterogeneous services; distributed data	Large optimization search space; coordination overhead
Organizational Gaps	Decentralized ownership; weak cost accountability; inconsistent tagging	Persistent spend growth; low attribution fidelity
Traditional Approaches	Manual cleanup; budget caps; ad hoc reviews	Operational risk; poor sustainability
Healthcare Constraints	Compliance; HA; auditability; PHI/PII protections	Narrow optimization envelope; strict guardrails
Root Causes	Poor utilization visibility, no automated remediation	Systemic inefficiencies

Table 2 *outlines inputs, actions, and expected outcomes by technique.*

Technique	Inputs	Action	Expected Outcome
Utilization-Based Rightsizing	Historical & real-time telemetry; SLOs	Adjust instance class/size via canary rollout	Higher utilization without SLO violations
Demand-Aware Autoscaling	Demand history; calendars; metrics	Predictive, scheduled, bounded scaling	Reduced off-peak waste; stable performance
Storage Lifecycle Optimization	Access patterns; retention; RTO/RPO	Automated tier transitions with	Lower unit storage cost at scale

		exceptions	
Commitment-Based Pricing	Baseline stability; growth forecasts	Apply and periodically adjust commitments	Sustained pricing discounts with guardrails
Idle Resource Remediation	Activity telemetry; tagging metadata	Detect and remove idle/orphaned assets	Budget recovery; regression prevention
Governance & Orchestration	Policies, cost & reliability signals, exceptions	Enforce guardrails via policy-as-code and pipelines	Durable optimization, auditability, regression prevention

Table 3 *summarizes the primary governance components, their implementation mechanisms, and enforcement methods.*

Section	Governance Mechanism	Implementation Detail	Enforcement / Outcome
Cost Ownership & Tagging	Mandatory resource metadata	Required tags (team, application, environment, cost center) validated in CI/CD	Non-compliant provisioning blocked; full cost attribution and drill-down analytics
Policy-as-Code Controls	Executable optimization standards	Autoscaling requirements, storage lifecycle policies, instance size bounds evaluated at deploy and runtime	Automated deny/remediate actions; drift detection prevents regressions
Cost Performance Observability	Correlated telemetry dashboards	Cost signals linked with availability, latency, throughput, and error rates	Early detection of adverse trade-offs; blameless post-optimization reviews
Operational Checkpoints	Embedded cost review gates	Standing checkpoints in team spend reviews and architecture governance; required cost impact notes	Efficiency treated as first-class concern in change management decisions

Table 4 *summarizes the public-safe, aggregate outcomes*

Metric	Year 1	Year 2
Annualized infrastructure cost reduction	≈18%	An additional ≈5%
Availability/latency adherence	Met SLOs	Met SLOs
Forecast variance vs. budget	Improved	Improved

12. Conclusions

Engineering-first cloud cost optimization, anchored in rightsizing, autoscaling, storage lifecycle management, commitment discipline, and policy-as-code can yield double-digit percentage savings initially and continued improvements thereafter, even as demand and total spend increase. By embedding cost checkpoints, observability, and ownership into routine operations, enterprises can sustain efficiency as a core reliability practice rather than a periodic financial intervention.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The author thanks the platform engineering, SRE, security, and finance partners who contributed to design, validation, and adoption. Any views expressed are those of the author and do not necessarily reflect the views of any employer or client.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

[1] K. Bennett et al., "State of the Cloud 2023," Bessemer Venture Partners, 2023. Available: https://www.bvp.com/atlas/state-of-the-cloud-2023

[2] Amazon Web Services, "COST 1. How do you implement cloud financial management?," AWS Well-Architected Framework, 2023. Available: https://docs.aws.amazon.com/wellarchitected/latest/framework/cost-01.html

[3] L. M. Dang et al., "A Survey on Internet of Things and Cloud Computing for Healthcare," Electronics, 8(7), 2019.

[4] W. Iqbal, M. Dailey, D. Carrera, and P. Janecek, "Adaptive resource provisioning for read-intensive multi-tier applications in the cloud," Future Generation Computer Systems, 27(6), 2011.

[5] N. Roy, A. Dubey, and A. Gokhale, "Efficient Autoscaling in the Cloud using Predictive Models for Workload Forecasting," Proc. IEEE CLOUD, 2011.

[6] A. Bhatnagar et al., "More for less: Five ways to lower cloud costs without destroying value," McKinsey, 2022. Available: https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/more-for-less-five-ways-to-lower-cloud-costs-without-destroying-value

[7] D. DeBellis and N. Harvey, "2023 State of DevOps Report: Culture is everything," Google Cloud, 2023. Available: https://cloud.google.com/blog/products/devops-sre/announcing-the-2023-state-of-devops-report