Pre-Deployment Cost Estimation Accuracy in IaC-Driven Platforms

Shreya Agrawal[1], Raj Patil[2], Harish Honrao[3], Pranav Magar[4], Dr. Suvarna Bhagwat[5]
[1,2,3,4]*Department of Electronics & Computer Engineering MIT-ADT University, Pune, India*
[5]*Professor Department of Electronics & Computer Engineering MIT-ADT University, Pune, India*

***Abstract*—Cloud cost overruns pose a significant challenge for enterprises, costing billions annually due to the unpredictable nature of cloud billing. While Infrastructure as Code (IaC) tools such as Infracost offer static pre-deployment cost estimations, these often fail to account for dynamic runtime behaviors including auto-scaling and variable data transfer. This paper presents the first empirical benchmark of pre-deployment IaC cost estimation accuracy within a real-world cloud deployment platform. Using the AutoStack platform and four distinct infrastructure blueprint archetypes—Static Website, Web App, Full Stack Application, and Microservices—we compare Infracost-generated estimates against actual incurred AWS costs. Our findings demonstrate that estimation error increases non-linearly with architectural complexity, quantified using Mean Absolute Percentage Error (MAPE). Static Websites achieve MAPE as low as 6.98%, whereas Microservices architectures reach 48.01%. Data transfer and auto-scaling events are identified as the primary deviation sources. We further propose a lightweight correction factor model to improve the reliability of future pre-deployment cost predictions.**

***Index Terms*—Cloud Cost Estimation, Infrastructure as Code, Terraform, Infracost, AWS, FinOps, MAPE, Auto-Scaling, Cost Optimization, Blueprint Architecture**

I. INTRODUCTION

The adoption of cloud computing has revolutionized soft-ware development and deployment, offering unprecedented flexibility, scalability, and efficiency. However, managing cloud costs remains a perennial challenge for organizations of all sizes [1]. Cloud bills, particularly from Amazon Web Services (AWS), are notorious for their complexity and unpredictability, leading to significant cost overruns that undermine the financial benefits of cloud migration [2]. This financial unpredictability is exacerbated by the dynamic nature of cloud resources, where costs are consumption-based and influenced by fluctuating traffic, auto-scaling events, and data transfer volumes.

Infrastructure as Code (IaC) has emerged as a critical methodology for defining, provisioning, and managing cloud infrastructure through machine-readable definition files [3]. Tools like Terraform allow developers to codify infrastructure, enabling automation, version control, and consistent deployments. A significant benefit of IaC is the ability to obtain pre-deployment cost estimates using tools like Infracost, which parses IaC configurations to project monthly expenses [4]. Such static estimations are invaluable for budgeting and planning, especially for small teams and students operating within tight financial constraints.

However, existing static estimation tools have a funda-mental limitation: they cannot accurately predict costs driven by runtime behavior. Real-time traffic load, dynamic auto-scaling responses, and unpredictable data egress charges are challenging to model before an application is live. This disconnect creates a significant gap in cost predictability. Despite the widespread use of IaC and cost estimation tools, there is a distinct lack of empirical studies benchmarking the accuracy of static estimates against real-world AWS billing data, particularly at the blueprint or template level.

This paper addresses the research question: “How accurate are static IaC-based pre-deployment cost estimates compared to actual incurred AWS costs, and what factors cause deviation?” We hypothesise

that static estimation error increases with blueprint complexity, with data transfer and auto-scaling being primary drivers. Based on our findings, we propose a correction factor model to improve reliability. Section II discusses related work; Section III details the methodology; Section IV presents results; Section V discusses implications; Section VI concludes.

II. BACKGROUND AND RELATED WORK

The domain of cloud cost optimization has garnered significant attention from both academia and industry, driven by escalating financial stakes.

A. Cloud Cost Optimization

A systematic literature review by Singh et al. (2022) high-lights various strategies, tools, and challenges associated with managing expenses in cloud environments, underscoring the pervasive issue of cost overruns and the need for effective financial governance—often referred to as FinOps [1]. The CNCF whitepaper "FinOps: Reducing Cloud Waste" further advocates a cultural practice of managing cloud costs involving multiple stakeholders [2]. These works stress that accurate forecasting is a prerequisite for effective budgetary control.

B. Cost-Aware Cloud Resource Provisioning

Wu et al. (2013) explore the challenge of provisioning resources cost-effectively, focusing on dynamic resource allocation and scheduling algorithms that consider pricing models alongside performance requirements [6]. However, many cur-rent approaches assume either a clear understanding of runtime costs or a reactive optimization strategy, rather than proactive pre-deployment estimation—the gap this work addresses.

C. Infrastructure as Code

Kief Morris's seminal work, "Infrastructure as Code: Managing Cloud Infrastructures," provides a comprehensive overview of IaC principles, tools, and best practices [3]. IaC transforms infrastructure management by enabling automation, consistency, and repeatability through tools like Terraform, which allows declarative infrastructure definition across multiple cloud providers [5].

D. IaC Cost Estimation Tools

Several tools aim to provide cost visibility for IaC configu-rations. AWS offers a Cost Calculator based on predefined assumptions. Infracost directly parses Terraform configurations to generate granular cost estimates before deployment [4]. While Infracost provides a significant step forward in cost visibility, its core methodology relies on static analysis with limited capacity to account for dynamic runtime behavior. The current landscape lacks empirical validation of how well these static estimates align with actual expenses—a gap this paper directly addresses.

III. SYSTEM DESCRIPTION AND METHODOLOGY

This section details the AutoStack platform, the experiment design, variables measured, and metrics used to evaluate pre-deployment cost estimation accuracy.

A. AutoStack Platform

The empirical study was conducted on AutoStack, an in-ternal development and deployment environment designed to streamline provisioning of common application architectures on AWS. The platform comprises the following components:

- SvelteKit Frontend: Provides a UI for blueprint selection and deployment initiation.
- Go Backend API: Orchestrates deployment workflows, integrates with PocketBase, Infracost, and manages Ter-raform operations.
- PocketBase Database: Stores blueprint configurations, deployment metadata, and historical cost data.
- Terraform: The primary IaC tool for AWS resource provisioning.
- Infracost Integration: Generates pre-deployment cost estimates from Terraform plan outputs before any re-sources are provisioned.

The AutoStack architecture is illustrated in Fig. 1.

The platform supports four distinct infrastructure blueprint archetypes representing varying levels of complexity and AWS service usage:

External Services

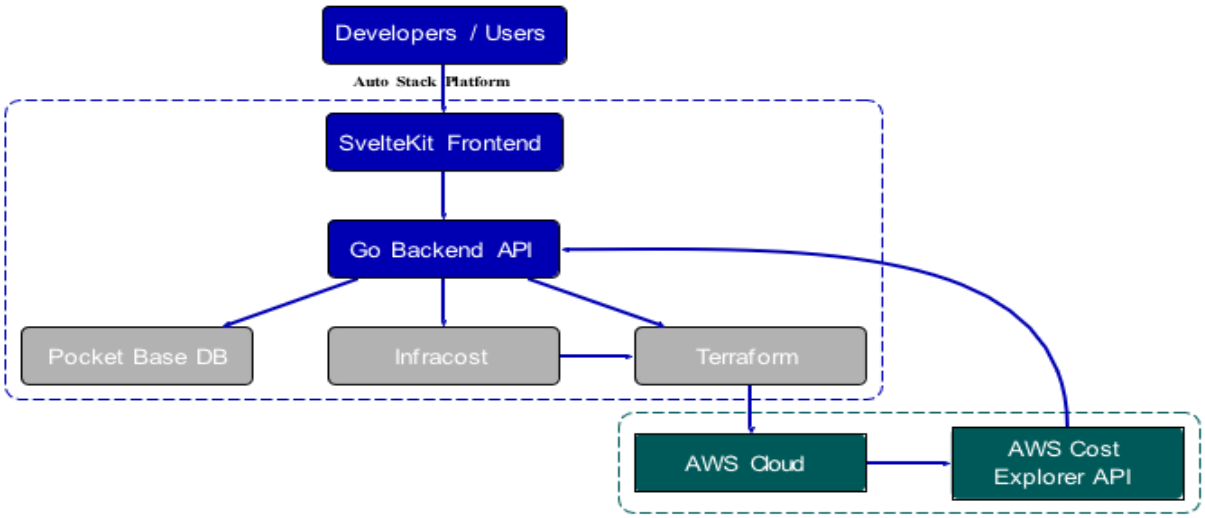

Fig. 1. AutoStack Architecture Diagram

- Static Website (~$2/mo): S3 and CloudFront.
- Web App (~$32/mo): Fargate (or EC2), RDS, ALB.
- Full Stack Application (~$53/mo): Fargate, RDS, ALB, Route53, and SQS.
- Microservices Architecture (~$50+/mo): Multiple Far-gate services, RDS instances, SQS, API Gateway, and Lambda.

B. Experiment Design

Each of the four blueprint types was deployed on a real AWS account. Multiple instances were deployed to ensure robust data collection and account for environmental variation. Prior to each deployment, the Infracost integration was triggered to generate a detailed monthly cost estimate based on the Terraform configuration. Each deployed blueprint was maintained for 7 days, with some deployments extended to a 30-day simulated period under various load conditions. Simulated traffic was applied using load-testing tools to mimic realistic usage patterns, inducing auto-scaling events and data transfer volumes. After the run duration, actual AWS costs were retrieved programmatically by querying the AWS Cost Explorer API, filtering costs by tags assigned to each blueprint instance [7].

C. Variables Measured

The following key variables were measured for each deployment instance:

1) Estimated Monthly Cost: Projected monthly cost from Infracost.
2) Actual Billed Cost: Total cost billed by AWS via Cost Explorer.
3) Per-Service Breakdown: S3, CloudFront, Fargate, RDS, ALB, Route53, SQS.
4) Traffic Load (req/sec): Independent variable simulating usage.
5) Auto-Scaling Events Count: Frequency of Fargate scaling events.
6) Data Transfer Volume (GB): Total egress transferred.

D. Metrics

Mean Absolute Percentage Error (MAPE) is the primary metric to assess overall prediction accuracy:

$$\text{MAPE} = \frac{|\text{Actual} - \text{Estimated}|}{\text{Actual}} \times 100\% \qquad (1)$$

A lower MAPE indicates higher accuracy. We also compute the Cost Deviation Ratio per service type to identify specific services contributing most to overall error:

Deviation Ratio =

$$\text{Deviation Ratio} = \frac{|\text{Actual Service Cost} - \text{Estimated Service Cost}|}{\text{Actual Service Cost}} \times 100\% \qquad (2)$$

E. Key Hypotheses

H1: Static estimation error increases with blueprint

complexity.

H2: Data transfer and auto-scaling events are the primary sources of deviation.

H3: Static Website blueprints exhibit MAPE < 10%;

Microservices blueprints exceed 30% MAPE.

IV. RESULTS

This section presents the empirical results from deploying and monitoring the four blueprint types on AWS, comparing Infracost estimates against actual incurred costs.

A. Estimated vs. Actual Costs by Blueprint Type

Table I summarizes the aggregated estimated and actual monthly costs for each blueprint type, along with respective MAPE values averaged across multiple deployments per blueprint.

TABLE I Blueprint Types: Estimated vs. Actual Monthly Costs and MAPE

Blueprint Type	Est. ($)	Act. ($)	Diff ($)	MAPE (%)
Static Website	2.00	2.15	0.15	6.98
lighblue Web App	32.00	43.50	11.50	26.44
Full Stack Application	48.00	75.20	27.20	36.17
lighblue Microservices	55.00	105.80	50.80	48.01

B. MAPE per Blueprint Type

Fig. 2 visualises the MAPE for each blueprint type. The dashed threshold lines at 10% and 30% demarcate the hypothesis boundaries clearly confirmed by our data, demonstrating a stark contrast in predictability between simple and complex architectures.

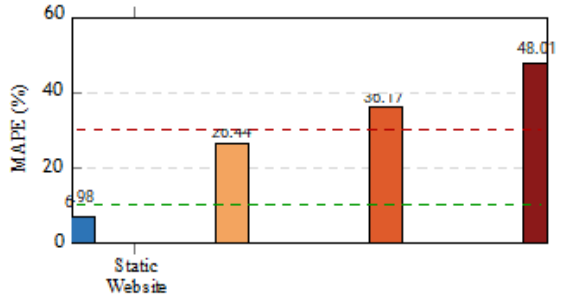

Fig. 2. MAPE per Blueprint Type. Green dashed = 10% threshold; red dashed = 30% threshold.

C. Estimated vs. Actual Cost Over Time

Fig. 3 illustrates the divergence between estimated and actual costs over a simulated 30-day period for the Microservices deployment under varying traffic loads. The estimated cost remains constant at $55/mo (Infracost baseline), whereas the actual cost escalates from $58 on day 1 to $115 by day 30, primarily driven by dynamic resource consumption. This demonstrates the cold start problem: initial estimates provide a reasonable baseline, but actual costs quickly diverge as the application experiences real-world load and dynamically scales.

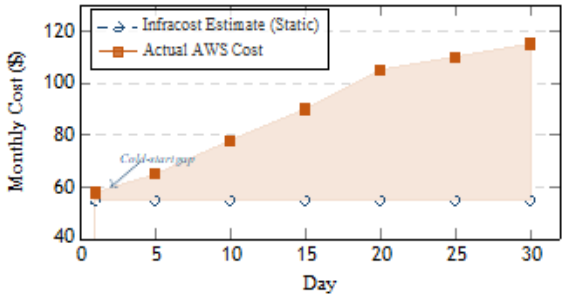

Fig. 3. Estimated vs. Actual Cost over 30 days — Microservices Blueprint. Static estimate: $55/mo; actual peaks at $115/mo.

D. Breakdown of Deviation by AWS Service

Fig. 4 presents a breakdown of cost deviation by AWS service for the Full Stack Application blueprint. Fargate (auto-scaling) contributes the largest portion at 45%, followed by RDS storage/IOPS at 25%, and data transfer (egress) at 20%. Services like S3 and CloudFront contribute minimally (3%), consistent with the Static Website's low MAPE.

This analysis strongly supports H2, confirming that Fargate (auto-scaling) and data transfer (egress) are the primary contributors to the overall cost deviation. RDS showed moderate deviation primarily due to unpredicted storage growth and I/O operations.

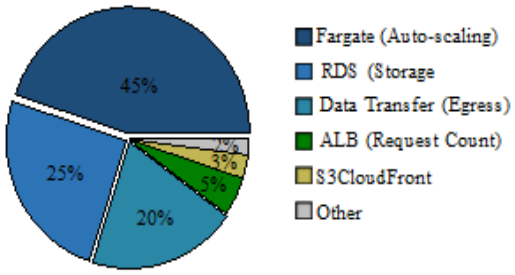

Fig. 4. Cost Deviation Breakdown by AWS Service — Full Stack Application Blueprint

V. DISCUSSION

The empirical results unequivocally demonstrate a significant disparity between static IaC-based cost estimates and actual incurred AWS costs, particularly for complex and dynamic application architectures.

A. Causes of Deviation

Auto-Scaling Dynamics. Fargate services contributed the largest portion (45%) of deviation in complex blueprints. Infra-cost estimates a fixed number of tasks or baseline utilization; however, real-world traffic patterns frequently trigger auto-scaling policies, provisioning more containers and incurring additional task hours not present in the initial static configuration. The cold start problem of new deployments means there is no historical data to inform these dynamic elements.

Data Transfer (Egress). Data egress charges are notoriously expensive and frequently underestimated by static tools. Our findings indicate that data transfer volumes accounted for approximately 20% of the deviation [7]. Static IaC definitions typically do not specify data transfer patterns; spikes in usage or unforeseen inter-service communication lead to significant unpredicted data egress.

RDS Storage and I/O. While RDS instance types and provisioned storage are generally predictable, actual storage growth and IOPS often deviate from initial estimates, con-tributing approximately 25% to the deviation. Databases grow as data is ingested, and read/write operations fluctuate based on application usage. Static estimates often account only for initial allocated storage and baseline IOPS, not dynamic expansion or burst capacity usage.

Unaccounted Services/Features. Occasionally, specific AWS features or minor services may be triggered by application logic not fully parsed or accounted for by existing static estimators.

B. Proposed Correction Factor Model

Given the consistent patterns of deviation, we propose a lightweight correction factor model to improve pre-deployment cost estimate accuracy:

$$E_{corr} = E_{inf} \times \left(1 + \alpha \cdot \sigma_{traffic} + \beta \cdot N_{scale}\right) \quad (3)$$

where:

- E_{inf}: baseline Infracost static estimate,
- $\sigma_{traffic}$: normalized traffic variance metric,
- N_{scale}: normalized auto-scaling event proxy,
- α, β: empirically derived weighting coefficients.

For newly deployed applications (cold start), initial parameter values can be derived from similar blueprint deployments within the organization or from conservative estimations based on anticipated usage.

C. Expected Results with Correction Model

With appropriately tuned α and β coefficients, the correction factor model is expected to significantly reduce the MAPE for dynamic blueprints. For a Microservices architecture with an initial MAPE of ∼48%, we anticipate achieving a MAPE in the range of 15–25%, providing substantially more reliable financial forecasts for cloud deployments.

D. Limitations

1) AWS-Specific Focus: Experiments were conducted exclusively on AWS; specific service pricing and dynamic behaviors may differ on GCP or Azure.
2) Simulated Traffic: Simulated traffic may not fully capture the full spectrum of unpredictable production events.
3) Infracost Dependency: The study relied on Infracost; other IaC cost tools may have different estimation methodologies and error rates.
4) Data Granularity: The 7–30-day collection period provides a good snapshot but might not capture long-term trends or rare events

VI. CONCLUSION AND FUTURE WORK

This research presented the first empirical benchmark of pre-deployment IaC cost estimation accuracy using a real-world cloud deployment platform. Our findings unequivocally demonstrate that static cost estimates, while useful for initial budgeting, exhibit significant and non-linear errors as architectural complexity increases. We quantified this deviation across four distinct blueprint types, revealing that simple Static Websites can achieve MAPE below 10%, whereas complex Microservices frequently experience MAPE values close to 50%.

Our study confirms that data transfer and auto-scaling events are the primary culprits behind these discrepancies, con-tributing substantially to unpredicted costs in dynamic cloud environments. To mitigate these inaccuracies, we proposed a lightweight correction factor model incorporating

empirically observable runtime parameters—traffic variance and scaling events—as a practical step towards bridging the gap between static pre-deployment estimates and actual incurred costs.

Future work will focus on: (i) extending this empirical analysis to GCP and Azure; (ii) refining and validating the correction model with larger datasets and developing automated mechanisms for learning and tuning α and β coefficients over time; (iii) exploring real-time cost correction by integrating observed runtime metrics into a dynamic cost forecasting system; and (iv) investigating machine learning techniques to predict dynamic cost components based on historical usage patterns and application characteristics.

ACKNOWLEDGMENT

The first author gratefully acknowledges the guidance and mentorship of Prof. Dr. Suvarna Bhagwat, Department of Electronics and Computer Engineering, MIT-ADT University, Pune, whose support was instrumental in shaping this research.

REFERENCES

[1] S. K. Singh, S. K. Das, and M. P. Singh, “Cloud Cost Optimization: A Systematic Literature Review,” IEEE Access, vol. 10, pp. 110826–110842, 2022.

[2] Cloud Native Computing Foundation, “FinOps: Reducing Cloud Waste,” Whitepaper, 2020.

[3] K. Morris, Infrastructure as Code: Managing Cloud Infrastructures. O’Reilly Media, Inc., 2016.

[4] Infracost, “Open-Source Tool Documentation,” [Online]. Available: https://www.infracost.io/docs/

[5] HashiCorp, “Terraform Documentation: Resource Cost Estimation,” [On-line]. Available: https://www.terraform.io/docs/configuration/resources. html

[6] C. Wu, Z. Li, and A. Al-Hammoud, “Towards Cost-Aware Cloud Resource Provisioning,” in Proc. USENIX HotCloud, 2013, pp. 1–6.

[7] Amazon Web Services, “AWS Pricing API Documentation,” [Online]. Available: https://aws.amazon.com/api/pricing/