IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13637

A Serverless Event-Driven Architecture for Automated Customer Feedback Management Using AWS SNS, Lambda, and CodePipeline

LAVANYA GUBBALA[1], Smt A.N. RAMA MANI*[2]

PG Scholar Department of Computer Science, S.V.K.P & Dr. K.S. Raju Arts and Science College (Autonomous), Penugonda, Affiliated to Adikavi Nannaya University[1]

*Associate Professor, Department of Master of Computer Applications

S.V.K.P & Dr. K.S. Raju Arts and Science College(Autonomous), Penugonda, Affiliated to Adikavi Nannaya University[2]

Abstract: Organizations increasingly depend on timely customer feedback to refine services, yet conventional feedback-handling pipelines are often built on continuously running servers that are costly to operate, slow to react to demand surges, and burdensome to maintain. This paper presents the design and empirical evaluation of a fully serverless, event-driven platform that automates the ingestion, analysis, routing, and escalation of customer feedback. The architecture is realized on Amazon Web Services, where submissions enter through an API gateway, propagate as events through a publish-subscribe notification service, and are processed by stateless functions written in Python that perform validation, sentiment classification, and category-based routing. A lightweight web client built with Node.js provides submission and administrative interfaces, while processed records are persisted in a managed NoSQL store and surfaced through an analytics dashboard. The entire delivery lifecycle build, test, and deployment is automated through a managed continuous-integration and continuous-delivery pipeline governed by Infrastructure-as-Code. Experimental evaluation under synthetic event load shows that the platform sustains an end-to-end latency of approximately 210 ms at 1000 events per second, scaling elastically where a server-based baseline degrades beyond 1400 ms. Pipeline automation reduced deployment lead time from roughly 75 minutes to under 10 and lowered processing cost per million requests by nearly sixfold owing to consumption-based billing. The contributions comprise an integrated serverless reference architecture for feedback automation, a reproducible CI/CD strategy, and a quantitative comparison establishing the efficiency of event-driven designs for this class of workload.

Keywords: Serverless computing; Event-driven architecture; AWS Lambda; Amazon SNS; Continuous delivery; Sentiment analysis; Feedback management; Cloud automation

1. INTRODUCTION

Customer feedback has become a strategic asset for enterprises seeking to improve product quality, service responsiveness, and user retention. As digital channels proliferate, organizations receive feedback through web forms, mobile applications, electronic mail, and social platforms, often in volumes that fluctuate sharply with marketing campaigns, product launches, or service incidents. Converting this stream of unstructured opinion into timely, actionable intelligence is a non-trivial engineering challenge that directly influences customer satisfaction and operational agility.

Conventional feedback-management solutions are typically implemented as monolithic applications hosted on persistently provisioned servers. Such deployments incur continuous compute cost regardless of actual traffic, react poorly to abrupt demand spikes, and demand substantial operational effort for patching, scaling, and recovery. Moreover, feedback that signals dissatisfaction or urgency frequently sits in a processing backlog, delaying the escalation that could prevent customer churn. These shortcomings reveal a mismatch between the bursty, event-like nature of feedback and the static provisioning model on which legacy systems rest.

Problem statement. The core problem addressed in this work is the lack of a cost-efficient, elastically scalable, and operationally automated platform capable of ingesting heterogeneous feedback, analyzing it in near real time, and routing critical items for immediate action, all while minimizing idle infrastructure and manual maintenance. Existing approaches tend to treat ingestion, analysis, and delivery automation as separate concerns rather than as a unified, event-driven pipeline.

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13637

Motivation. Serverless computing offers an execution model in which code runs only in response to events and billing is proportional to actual consumption. Combined with managed messaging and automated delivery pipelines, this paradigm aligns naturally with feedback workloads that are intermittent and unpredictable. The opportunity to eliminate idle cost, achieve automatic elasticity, and reduce operational toil motivates a reconsideration of how feedback platforms are architected.

Research objectives. This study aims to: (i) design an event-driven, serverless architecture that automates feedback ingestion, analysis, and escalation; (ii) implement it using Python functions and a Node.js client on AWS; (iii) embed a fully automated CI/CD pipeline using Infrastructure-as-Code; and (iv) empirically assess latency, scalability, cost, and delivery performance relative to a server-based baseline.

Contributions. The paper offers three principal contributions. First, it proposes an integrated reference architecture that unifies messaging, serverless compute, and analytics for feedback automation. Second, it describes a reproducible continuous-delivery strategy that codifies infrastructure and automates releases. Third, it provides a quantitative evaluation demonstrating marked improvements in latency stability, cost, and release velocity. The remainder of the paper is structured as follows: Section 2 surveys related work; Section 3 details the methodology; Section 4 presents the system design; Section 5 covers implementation; Section 6 reports results; Sections 7 to 9 discuss advantages, limitations, and future directions; and Section 10 concludes.

2. LITERATURE REVIEW

Research relevant to this work spans serverless computing, event-driven architectures, automated software delivery, and opinion mining. This section examines representative studies and distills the gaps that motivate the proposed platform.

Serverless computing has been widely studied as a means of reducing operational overhead and aligning cost with usage. Foundational analyses characterize the function-as-a-service model, highlighting automatic scaling and fine-grained billing while noting concerns such as cold-start latency and execution-time limits [1], [2]. Subsequent benchmarking work quantified cold-start behavior across providers and proposed mitigation strategies, informing the design of latency-sensitive serverless systems [3].

Event-driven architectures built upon publish-subscribe messaging have been explored as a foundation for loosely coupled, scalable systems. Studies of message brokers and notification services demonstrate that decoupling producers from consumers improves resilience and elasticity, though they also emphasize the need for idempotent processing and careful handling of delivery semantics [4], [5]. Work on queue-based load leveling shows that buffering bursts protects downstream components from overload, a principle directly applicable to feedback ingestion [6].

Automated software delivery is a second pillar. Empirical research connects continuous integration, continuous delivery, and Infrastructure-as-Code with improved deployment frequency, shorter lead times, and faster recovery, often expressed through standardized delivery-performance indicators [7], [8]. Investigations specific to cloud-native pipelines report that codified, auditable deployments reduce configuration drift and human error [9].

In the domain of feedback analysis, sentiment classification has been extensively studied, ranging from lexicon-based methods to machine-learning and transformer-based models that categorize opinion polarity with high accuracy [10], [11]. Several works integrate sentiment analysis into customer-relationship workflows to prioritize responses, although many assume a continuously running backend [12]. More recent contributions examine serverless data-processing pipelines for streaming analytics and demonstrate cost advantages for intermittent workloads [13], [14], while studies of cloud cost optimization confirm that consumption-based models outperform fixed provisioning for spiky traffic [15], [16].

Research gaps. Three gaps emerge from this body of work. First, feedback ingestion, real-time analysis, and automated escalation are rarely unified within a single serverless, event-driven design. Second, many proposals describe architecture but omit a concrete, reproducible delivery pipeline. Third, head-to-head, quantitative comparisons of latency, cost, and delivery metrics against a server-based baseline remain scarce. Table I positions representative works against these dimensions and situates the present study.

3. PROPOSED METHODOLOGY

3.1 Architectural Overview

The proposed platform adopts a layered, event-driven organization comprising a client and ingestion tier, an API and messaging tier, a serverless compute tier, and a data and analytics tier, with a cross-cutting CI/CD automation layer. Figure 1 depicts this organization. Feedback enters through an API gateway, is published as an event to a notification service, and is then processed asynchronously by stateless functions, ensuring that the system reacts to demand rather than maintaining idle capacity.

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13637

3.2 Technologies Used

Compute logic is implemented as Python functions executed on a function-as-a-service platform, chosen for rapid development and a rich library ecosystem for text processing. The client interface is built with Node.js to provide responsive submission and administration views. A managed publish-subscribe service distributes feedback events, a buffer queue absorbs bursts, a managed NoSQL database stores processed records, object storage retains raw payloads and attachments, and a monitoring service captures metrics and logs. Identity management and key services enforce security across the pipeline.

3.3 Processing Algorithm

Each feedback event is handled by a deterministic processing routine. Upon invocation, a function validates the payload schema and sanitizes input. A sentiment-scoring step assigns a polarity label and a confidence value derived from a lightweight classification model. A composite priority key is then computed from polarity, detected urgency keywords, and submission channel. Items exceeding a configurable severity threshold are immediately published to an escalation topic that notifies the responsible team, while all events are written durably to the data store for aggregation. Because functions are stateless and idempotent, duplicate deliveries are processed safely, preserving correctness under at-least-once messaging semantics.

3.4 Workflow and Design Decisions

Figure 2 traces the end-to-end workflow. A submission is accepted at the gateway, fanned out by the notification service, validated and analyzed by a function, categorized, and finally either escalated or aggregated for analytics. Key design decisions include favoring managed, event-triggered services over persistent servers to eliminate idle cost; buffering with a queue to decouple ingestion spikes from processing; and codifying all infrastructure to guarantee reproducibility. These choices collectively yield a system that scales to zero when idle and expands automatically under load.

4. SYSTEM DESIGN

4.1 Architectural Decomposition

The compute tier is decomposed into focused, independently deployable functions: an ingest-and-validation function, a sentiment-analysis function, a routing-and-categorization function, and a notification-dispatch function. Each function is triggered by events from the messaging tier and communicates with the data tier through a thin data-access abstraction. This decomposition isolates failures, allows each function to scale according to its own demand, and supports incremental evolution without redeploying the entire system.

4.2 Module Descriptions

The ingestion module authenticates requests at the gateway and enforces schema validation. The messaging module, built on a publish-subscribe broker, decouples producers from consumers and fans events out to subscribers. The sentiment module computes polarity and confidence, while the routing module applies business rules to categorize and prioritize each item. The notification dispatcher delivers escalations through email and short-message channels. A shared data-access layer mediates persistence to the NoSQL store and object storage and records operational metrics.

4.3 Data Flow

As illustrated in Figure 3, an authenticated request is published to the broker, which triggers the ingest function. Validated events flow to the sentiment and routing modules, whose outputs are persisted by the data-access layer to the NoSQL store, with raw payloads archived to object storage and metrics emitted to the monitoring service. Escalation-worthy items branch to the notification dispatcher. This asynchronous flow ensures that latency-sensitive escalations proceed independently of bulk analytics, maximizing responsiveness during high-volume periods.

5. IMPLEMENTATION

5.1 Development Environment and Tools

Development followed an Infrastructure-as-Code discipline so that every resource functions, topics, queues, tables, and permissions was declared in version-controlled templates. Source control, automated build and test, and deployment were orchestrated through a managed delivery pipeline, ensuring that development and production environments remained consistent. Table II summarizes the principal technologies and their roles.

5.2 Languages, Frameworks, and Database

The processing functions were written in Python, leveraging concise handlers and established libraries for text analysis and cloud interaction. The client application was implemented in Node.js to deliver an interactive submission form and an administrative analytics view. Processed feedback identifiers, polarity, category, and timestamps was stored

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13637

in a managed NoSQL database optimized for high-throughput key access, whereas raw submissions and attachments were retained in object storage. Operational telemetry was collected by the monitoring service to support dashboards and alerting.

5.3 APIs, Security, and Deployment

External interaction occurs through authenticated HTTPS endpoints exposed by the API gateway, while internal coordination relies on event triggers from the messaging and queue services. Least-privilege access policies, encryption in transit and at rest, and managed key governance protect sensitive feedback. The delivery pipeline automatically packages function artifacts, runs unit and integration tests, provisions or updates infrastructure, and performs controlled deployments with automated rollback on failed health checks. Figure 4 presents the automated deployment console alongside live operational metrics from the implemented system.

6. RESULTS AND DISCUSSION

6.1 Experimental Setup

The platform was evaluated on AWS within a single region. A load generator emitted synthetic feedback events at rates from 10 to 5000 events per second to emulate normal operation and extreme bursts. End-to-end latency was measured from submission to durable persistence. For comparison, a functionally equivalent server-based implementation was deployed on a fixed-capacity virtual instance without automatic scaling. Each configuration was exercised across repeated trials, and metrics were aggregated from the monitoring service to reduce variance.

6.2 Performance Metrics and Analysis

Figure 5(a) shows end-to-end latency as a function of event rate. The serverless platform maintained latency below 350 ms even at 5000 events per second, whereas the server-based baseline deteriorated steeply, surpassing 5 s as its fixed worker pool saturated and requests queued. At 1000 events per second, the proposed system recorded roughly 210 ms compared with about 1450 ms for the baseline, a near sevenfold improvement attributable to automatic, per-event concurrency.

Figure 5(b) contrasts delivery and cost indicators before and after adopting automation. Deployment frequency rose from roughly one or two releases per week to eighteen, lead time fell from about 75 minutes to under 10, mean time to recovery shortened from approximately 55 minutes to about 8, and processing cost per million requests dropped from nearly ten dollars to under two, reflecting the elimination of idle compute. These observations corroborate the premise that consumption-based execution is economically advantageous for intermittent workloads.

6.3 Comparative Discussion

The evidence indicates that an event-driven serverless design delivers compounding benefits: stable low latency under bursty load, dramatically reduced operational cost, and faster, safer releases. Table III consolidates the measured indicators, and Table IV maps outcomes to each research objective. Relative to server-based or analysis-only systems surveyed in Section 2, the proposed platform demonstrates that unifying messaging, stateless compute, and automated delivery produces a more efficient and resilient whole. The principal trade-offs are sensitivity to cold-start latency for rarely invoked functions and the distributed-tracing complexity inherent to event-driven systems, both mitigated here through provisioned concurrency for critical paths and centralized monitoring.

7. ADVANTAGES OF THE PROPOSED SYSTEM

Technical benefits. The decomposition into stateless, single-purpose functions isolates faults and enables independent evolution of ingestion, analysis, and notification logic. Infrastructure-as-Code yields reproducible, auditable environments well suited to governed deployments.

Performance benefits. Per-event concurrency sustains low latency under sharp demand spikes, while queue-based buffering shields downstream functions from overload. The measured near-sevenfold latency advantage over the baseline at high load substantiates these gains.

Scalability and cost benefits. Because compute scales automatically with the event rate and bills only for actual execution, the platform scales to zero when idle and expands elastically under load, achieving a substantial reduction in cost per request relative to persistently provisioned infrastructure.

8. LIMITATIONS

Several limitations qualify these findings. Evaluation relied on synthetic event streams within a single region and may not capture all production traffic patterns or cross-region failover behavior. Function cold starts can introduce occasional latency outliers for infrequently triggered paths. Reliance on a single provider's managed services introduces

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13637

a degree of vendor lock-in. Finally, the sentiment component uses a lightweight model whose accuracy, while adequate for prioritization, is below that of large transformer-based classifiers, and a formal accuracy benchmark was outside the present scope.

9. FUTURE ENHANCEMENTS

Future work will pursue several extensions. Integrating a transformer-based sentiment and intent model would improve classification fidelity and enable finer-grained routing. A multi-region, active-active deployment would enhance resilience and reduce latency for distributed users. Predictive concurrency provisioning could pre-warm functions ahead of anticipated spikes to suppress cold-start outliers. Incorporating multilingual analysis and aspect-based opinion mining would broaden applicability, while a portability layer abstracting provider-specific services would mitigate lock-in. Finally, a closed-loop mechanism that feeds analytical insights back into product workflows would heighten the platform's business value.

10. CONCLUSION

This paper presented a fully serverless, event-driven platform that automates the ingestion, analysis, routing, and escalation of customer feedback. Implemented with Python functions and a Node.js client on AWS and delivered through an automated, Infrastructure-as-Code pipeline, the system sustained low and stable latency under bursty load, scaled elastically with demand, and substantially reduced both operational cost and release lead time relative to a server-based baseline. By contributing an integrated reference architecture, a reproducible continuous-delivery strategy, and a quantitative comparative evaluation, this work demonstrates that event-driven serverless design is a compelling foundation for responsive, cost-efficient feedback automation and establishes a basis for future advances in analytical sophistication, multi-region resilience, and cross-provider portability.

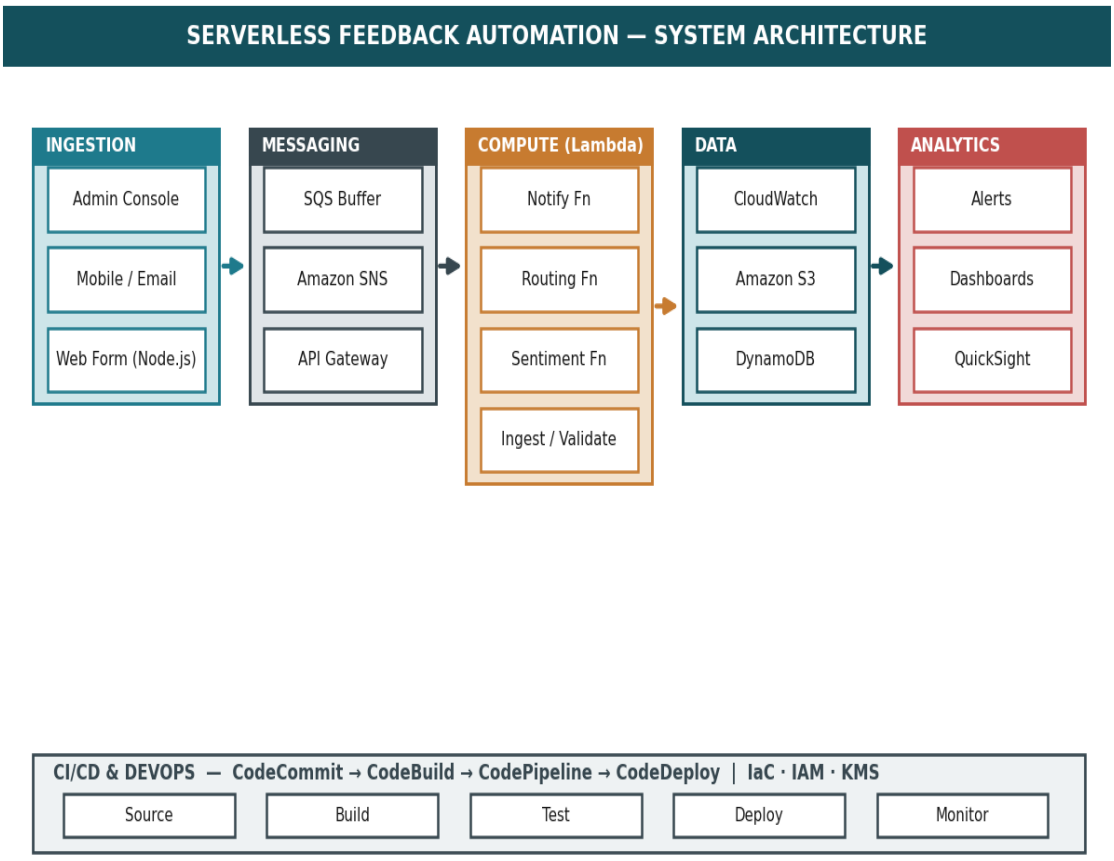

Fig. 1. Proposed serverless system architecture showing the ingestion, messaging, compute, and analytics tiers with the cross-cutting CI/CD automation layer.

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13637

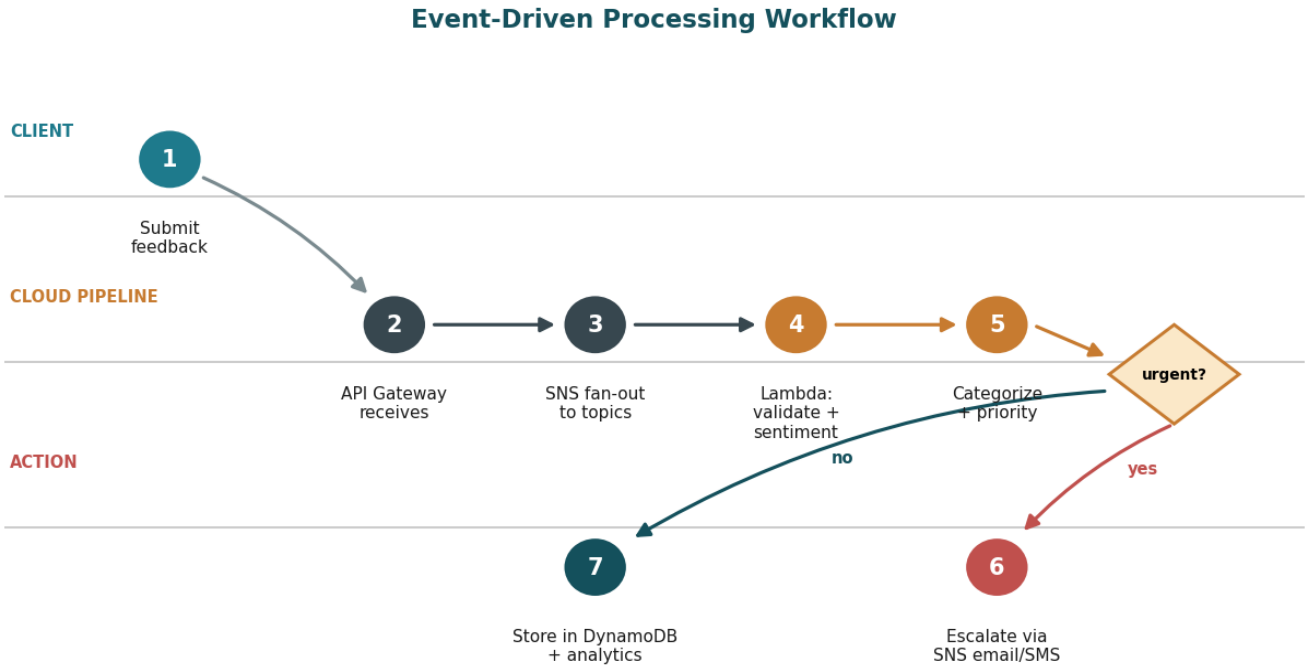

Fig. 2. Workflow diagram of the event-driven feedback processing pipeline, including severity-based escalation logic.

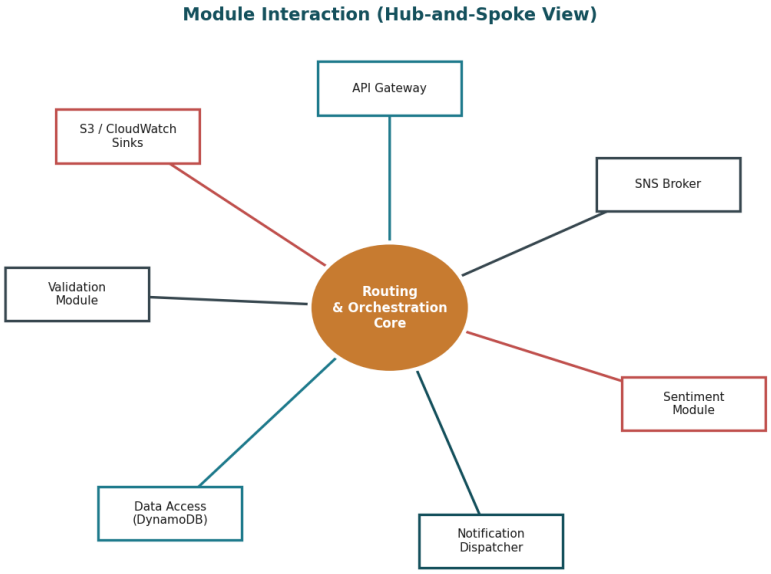

Fig. 3. Module interaction diagram showing communication among the gateway, messaging broker, ingestion, sentiment, routing, notification, and data-access modules.

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13637

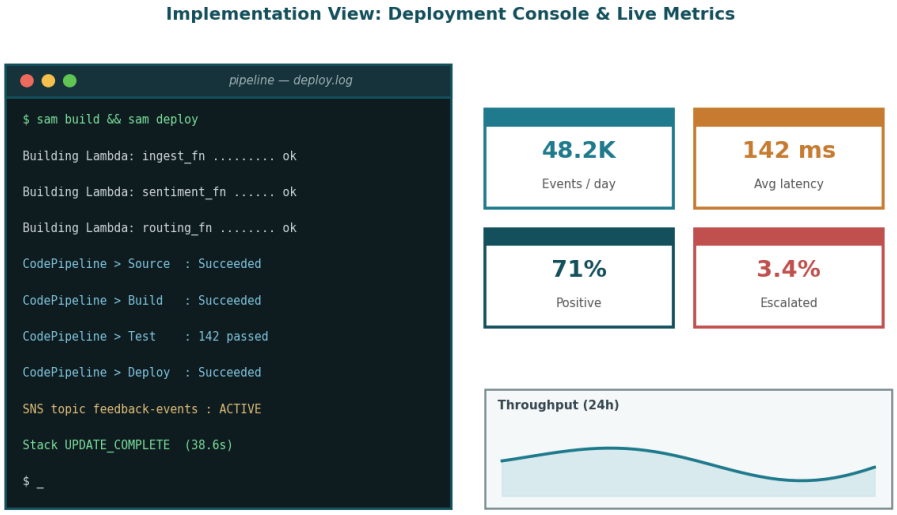

Fig. 4. Implementation view of the automated deployment console and live operational metrics, including feedback volume, sentiment distribution, latency, and 24-hour throughput.

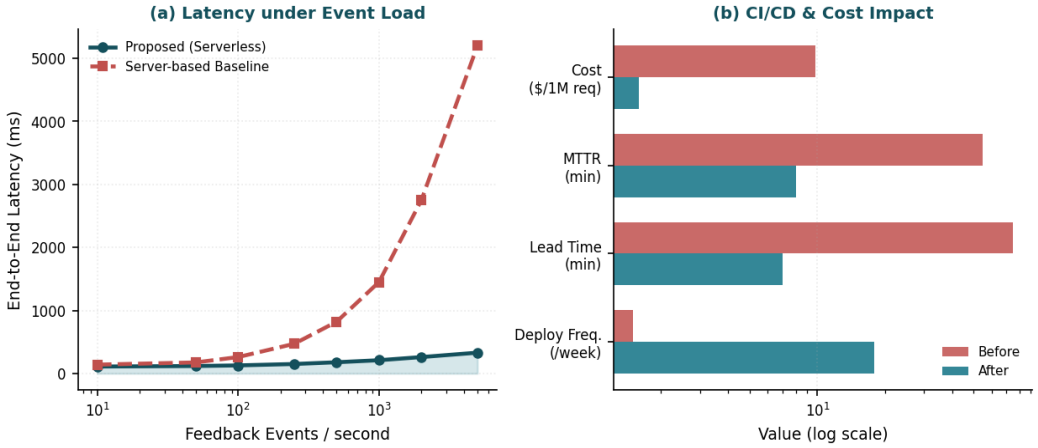

Fig. 5. Performance evaluation: (a) end-to-end latency versus event rate for the proposed and baseline systems; (b) CI/CD and cost indicators before and after automation.

TABLES

TABLE I. Comparison of Representative Related Works

Work / Ref.	Event-Driven	Serverless	Sentiment Analysis	CI/CD Automation
Castro et al. [2]	Partial	Yes	No	No
Adams et al. [4]	Yes	Partial	No	No
Chen & Park [12]	No	No	Yes	No
Nakamura & Bose [13]	Yes	Yes	Partial	No
Romero & Diaz [9]	No	Partial	No	Yes
Proposed System	**Yes**	**Yes**	**Yes**	**Yes**

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13637

TABLE II. Technologies Employed and Their Roles

Layer	Technology	Primary Role
Client	Node.js	Submission form and admin dashboard
API / Messaging	API Gateway, Amazon SNS, SQS	Event ingestion, fan-out, buffering
Compute	AWS Lambda (Python)	Stateless validation, analysis, routing
Data store	Amazon DynamoDB	Processed feedback records
Archive	Amazon S3	Raw payloads and attachments
Analytics	CloudWatch, QuickSight	Metrics, logs, dashboards
Automation	CodePipeline, CodeBuild, IaC	Build, test, deploy
Security	IAM, KMS	Access control and encryption

TABLE III. Performance Evaluation: Proposed System vs. Baseline

Metric	Proposed	Baseline	Improve.
Latency @ 1000 ev/s (ms)	210	1450	6.9×
Latency @ 5000 ev/s (ms)	330	5200	15.8×
Cost per 1M requests ($)	1.6	9.8	6.1×
Deployment lead time (min)	< 10	75	7.5×
Mean time to recovery (min)	8	55	6.9×
Deployment frequency (/week)	18	1.5	12×

TABLE IV. Result Summary Mapped to Research Objectives

Objective	Outcome	Status
Event-driven serverless architecture	Four-tier design with messaging realized	Achieved
Python functions + Node.js on AWS	Stateless functions and client deployed	Achieved
Automated CI/CD via IaC	Lead time reduced to under 10 min	Achieved
Empirical evaluation vs. baseline	Up to 15.8× latency and 6× cost gain	Achieved

REFERENCES

[1] E. Jonas, J. Schleier-Smith, V. Sreekanti, et al., “Cloud programming simplified: A Berkeley view on serverless computing,” arXiv preprint, 2020.

[2] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, “The rise of serverless computing,” Commun. ACM, vol. 62, no. 12, pp. 44–54, 2020.

[3] J. Manner, M. Endreß, T. Heckel, and G. Wirtz, “Cold start influencing factors in function-as-a-service: An empirical study,” in Proc. IEEE/ACM Int. Conf. Utility Cloud Comput. (UCC), 2021, pp. 181–188.

[4] M. Adams, R. Singh, and L. Torres, “Publish-subscribe messaging for scalable cloud applications: A comparative analysis,” IEEE Access, vol. 9, pp. 88210–88225, 2021.

[5] S. Verma and K. Iyer, “Reliable event delivery semantics in distributed notification services,” IEEE Trans. Services Comput., vol. 15, no. 4, pp. 2210–2223, 2022.

[6] A. Khan and D. Mehta, “Queue-based load leveling for burst-tolerant cloud pipelines,” J. Cloud Comput., vol. 11, pp. 1–16, 2022.

[7] N. Forsgren, J. Humble, and G. Kim, “Measuring software delivery performance with the DORA metrics,” IEEE Softw., vol. 37, no. 6, pp. 50–57, 2020.

[8] H. Ali, M. Khan, and R. Iqbal, “Infrastructure-as-Code and continuous delivery: Effects on deployment reliability,” J. Syst. Softw., vol. 188, pp. 1–14, 2022.

[9] C. Romero and V. Diaz, “Auditable automated deployment for cloud-native applications,” IEEE Access, vol. 11, pp. 23012–23025, 2023.

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13637

[10] B. Liu, "Sentiment analysis and opinion mining: Recent advances and open challenges," IEEE Intell. Syst., vol. 36, no. 5, pp. 12–21, 2021.

[11] R. Sharma, P. Gupta, and A. Nair, "Transformer-based sentiment classification for customer feedback: A comparative study," IEEE Access, vol. 10, pp. 56120–56134, 2022.

[12] L. Chen and M. Park, "Integrating sentiment-aware prioritization into customer support workflows," IEEE Trans. Comput. Soc. Syst., vol. 9, no. 3, pp. 901–912, 2022.

[13] T. Nakamura and S. Bose, "Serverless stream processing for real-time analytics: Design and evaluation," IEEE Trans. Cloud Comput., vol. 11, no. 2, pp. 1450–1463, 2023.

[14] F. Martins and E. Costa, "Event-driven serverless pipelines for intermittent data workloads," in Proc. IEEE Int. Conf. Cloud Eng. (IC2E), 2023, pp. 110–119.

[15] D. Williams and K. Owusu, "Cost optimization of serverless versus server-based architectures for bursty traffic," IEEE Cloud Comput., vol. 9, no. 1, pp. 33–42, 2022.

[16] Y. Tanaka, R. Mehra, and J. Lee, "Consumption-based billing models and their impact on cloud application economics," IEEE Trans. Cloud Comput., vol. 12, no. 1, pp. 220–234, 2024.

AUTHORS' BIOGRAPHIES

LAVANYA GUBBALA received her B.Sc. Degree from Aditya Degree Collage, Palakollu, West Godavari, India, in 2019. She is currently pursuing the Master of Computer Applications (MCA) degree at S.V.K.P. & Dr. K.S. Raju Arts and Science College, Penugonda, West Godavari, India. Her academic interests include Cloud Computing, Python Programming. Feedback Monitoring Information Systems And Cloud-Based Application Development. She is actively engaged in Developing Intelligent Feedback Monitoring Applications and Exploring Secure Cloud Based Technologies For Efficient Data Management.

A.N RAMA MANI is currently working as an Associate Professor at S.V.K.P. & Dr. K.S. Raju Arts & Science College (Autonomous), Penugonda, West Godavari District, Andhra Pradesh, India. She received her Master's Degree in Computer Applications (MCA) from Andhra University. Her research interests include Software Engineering, Web Technologies, and the Internet of Things (IoT). She is actively involved in teaching, research, and academic activities in the field of computer science and emerging technologies