IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 6, June 2026

DOI: 10.17148/IJARCCE.2026.15694

A Serverless Event-Driven Framework for Scalable Subscription Billing Using Function-as-a-Service and Automated Build Pipelines

Vasamsetti. Komalika[1], K. Lakshmi Sai Sri*[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]

Lecturer, 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: The proliferation of subscription-based commerce has intensified the demand for billing systems that can process recurring charges, prorations, and usage-based fees reliably at fluctuating scale. Traditional billing platforms are typically deployed on continuously provisioned servers, which incur idle cost during quiescent periods, scale coarsely under sudden demand, and entangle billing logic with infrastructure management. This study proposes a serverless, event-driven framework that administers the complete subscription billing cycle through stateless functions invoked on demand. The methodology decomposes billing concerns—subscription management, invoice generation, payment processing, usage metering, and notification—into independent functions exposed through a managed application programming interface gateway and coordinated by event and schedule triggers. Business logic is implemented in Python, while a Node.js client layer mediates user interaction, and an automated build service compiles, tests, packages, and deploys function revisions without manual provisioning. An experimental evaluation conducted across concurrency levels from fifty to thirty-two hundred simultaneous requests demonstrates that the serverless framework sustains an average latency of approximately 140 milliseconds at moderate load and degrades only gradually under heavy concurrency, whereas a provisioned-server baseline exhibits steep latency growth beyond four hundred concurrent requests. The framework attains a scaling efficiency near 94 percent and reduces cost per million requests by more than fourfold relative to the baseline, at the expense of occasional cold-start latency. The principal contributions comprise an event-driven billing architecture, a pipeline-automated deployment model for functions, and an empirical cost-performance analysis validating function-as-a-service for elastic billing workloads.

Keywords: Serverless computing, function-as-a-service, subscription billing, event-driven architecture, API gateway, continuous deployment, scalability, cloud cost optimization.

1. INTRODUCTION

Subscription commerce has become a dominant model across software, media, and digital services, in which providers levy recurring charges in exchange for continued access. Administering this model requires a billing system capable of managing subscription plans, computing prorated and usage-based charges, orchestrating payment collection, handling failures and retries, and notifying customers, all while maintaining accurate financial records. As subscriber bases expand and billing cycles concentrate activity into periodic bursts, the underlying system must accommodate highly variable load with both reliability and cost efficiency [1], [2].

Conventional billing platforms are commonly hosted on persistently running servers sized for peak demand. This approach wastes resources during the long intervals between billing runs, scales in coarse increments that lag sudden surges, and couples financial logic tightly to infrastructure operation. Capacity planning becomes difficult when demand is spiky, and the operational burden of patching, scaling, and maintaining always-on servers diverts effort from the billing logic itself [3], [4]. These inefficiencies motivate an execution model that allocates compute precisely when needed.

A. Problem Statement

There is a need for a subscription billing system that scales automatically and instantaneously with demand, charges only for compute actually consumed, and isolates individual billing concerns so that failures and updates remain localized.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 6, June 2026

DOI: 10.17148/IJARCCE.2026.15694

Existing server-bound architectures rarely satisfy these requirements simultaneously, leaving a gap between elastic, cost-proportional execution and reliable recurring-billing operation.

B. Motivation and Objectives

Motivated by these limitations, this work designs and evaluates a serverless framework. The objectives are to: (i) decompose the billing cycle into independent stateless functions triggered by events and schedules; (ii) expose these functions through a managed API gateway with integrated authentication; (iii) automate build, test, and deployment of function revisions through a managed build pipeline; and (iv) empirically quantify latency, scalability, and cost relative to a provisioned-server baseline.

C. Contributions

- An event-driven serverless architecture that administers the full subscription billing cycle through on-demand stateless functions.
- A decomposition of billing concerns into independently deployable functions, enabling fine-grained scaling and fault isolation.
- A pipeline-automated deployment model that builds, tests, and releases function revisions without manual server provisioning.
- An empirical cost-performance evaluation demonstrating superior elasticity and cost efficiency over a server-bound baseline.

2. LITERATURE REVIEW

The proposed framework builds upon research in serverless computing, event-driven architecture, automated deployment, and digital billing systems. This section surveys representative works and identifies the gaps that motivate the design.

Foundational analyses of serverless computing [1], [5] characterize the function-as-a-service paradigm, in which fine-grained, stateless functions execute in response to events and are billed per invocation. These studies report strong elasticity and cost proportionality but also identify cold-start latency as a principal limitation. Subsequent investigations [6] propose techniques to mitigate cold starts through provisioned concurrency and runtime optimization, informing the present design's handling of latency-sensitive operations.

Event-driven architecture has been examined as a natural complement to serverless execution. Research on event coordination and messaging [7], [8] demonstrates that decoupling producers and consumers through queues and event buses improves resilience and scalability. Studies of API gateways [9] establish their role in routing, authentication, and throttling for function backends, which the proposed framework adopts as its entry point.

A considerable body of work addresses automated software delivery. Analyses of build-and-deploy pipelines [10], [11] report reduced release latency and defect rates, while infrastructure-automation studies [12] emphasize reproducible provisioning of cloud resources. These findings underpin the framework's pipeline-driven deployment of functions and gateway stages.

Within billing and financial systems, research on recurring-payment processing [2], [13] highlights requirements for idempotency, retry handling, and auditability, and studies of usage-based metering [14] address accurate consumption tracking. Comparative work on cloud cost models [15], [16] confirms that pay-per-use execution can substantially reduce expenditure for intermittent workloads, though it cautions that high-throughput sustained workloads may favor provisioned capacity.

Synthesizing these contributions reveals a clear gap: although serverless execution, event-driven coordination, and automated delivery are individually mature, their unified application to recurring subscription billing—with explicit attention to cost, elasticity, and reliability—remains underexplored. The present study addresses this gap, as summarized in Table I.

3. PROPOSED METHODOLOGY

A. Architectural Overview

The framework adopts a serverless, event-driven architecture, illustrated in Figure 1. Client requests from a web or mobile interface traverse a managed API gateway that performs authentication, request validation, and routing before invoking

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 6, June 2026

DOI: 10.17148/IJARCCE.2026.15694

the appropriate stateless function. Distinct functions handle subscription management, billing and invoicing, payment webhooks, usage metering, and notification, each executing independently and scaling automatically with the volume of incoming events.

Recurring billing runs are initiated by a scheduled trigger that invokes the billing functions at defined intervals, while asynchronous interactions—such as payment confirmations and downstream notifications—are mediated through an event bus and queue. Persistent state, including subscriptions, invoices, and ledgers, resides in a managed database, ensuring that the functions themselves remain stateless and horizontally scalable.

B. Core Procedures and Algorithms

Two procedural mechanisms are central. The first is a billing-computation procedure that, for each active subscription, determines the applicable plan, computes prorated and usage-based charges for the period, applies any adjustments, and generates an invoice. The procedure is designed to be idempotent so that repeated invocation—arising from retries or duplicate events—does not produce duplicate charges, an essential property in distributed, event-driven execution.

The second is an automated deployment procedure in which a source commit triggers a managed build service that compiles dependencies, executes automated tests, packages each function, and deploys the new revisions together with the corresponding API gateway stage. Only revisions passing automated verification are promoted, reducing release risk and eliminating manual server provisioning.

C. Technology Stack and Design Rationale

Function logic is implemented in Python, chosen for its expressiveness and strong support for data and financial computation, while a Node.js layer mediates client interaction with its efficient asynchronous I/O model. The API gateway provides a secure, managed entry point; the build service automates packaging and deployment; and managed database, queue, and event-bus services supply persistence and asynchronous coordination. Pairing stateless functions with managed backing services reflects a deliberate decision to maximize elasticity and minimize operational overhead, allocating compute only when billing events occur.

4. SYSTEM DESIGN

A. Operational Workflow

Figure 2 presents the billing workflow as a set of swimlanes spanning the user, the API and function layer, the billing engine, and payment and notification services. A subscriber selects a plan, whereupon the subscription function validates and records the subscription. At each billing cycle the engine applies the plan and proration logic, an invoice is generated, and the payment function charges the customer through an external gateway. Successful and failed outcomes are recorded in the ledger, and the notification function informs the customer, initiating retries when a charge fails. This separation of responsibilities across lanes clarifies the flow of control and the points of asynchronous handoff.

B. Module Descriptions

The platform comprises five principal modules whose interactions appear in Figure 3. The subscription module manages plans, sign-ups, upgrades, and cancellations. The billing and invoice module computes charges and produces invoices. The payment module integrates with external gateways and handles confirmations and retries. The usage-metering module records consumption for usage-based plans. The notification module dispatches receipts, reminders, and failure alerts. A persistence layer maintains subscriptions, invoices, and an immutable ledger, while an event bus coordinates asynchronous interactions; the entire collection is deployed as functions through the automated build target.

This decomposition ensures separation of concerns and fault isolation: a failure in payment processing does not impede subscription management, and each function scales according to its own invocation rate, which is advantageous when billing-cycle bursts load the invoicing function far more heavily than routine subscription changes.

5. IMPLEMENTATION

The prototype was developed and deployed within a cloud environment; its stack and tooling are summarized in Table II. Billing and server-side logic were authored in Python and packaged as discrete functions, while the client interface was built upon the Node.js runtime using a component-based single-page design to present plans, invoices, customers, and usage. The managed API gateway exposed secure HTTPS endpoints with integrated authentication and request validation.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 6, June 2026
DOI: 10.17148/IJARCCE.2026.15694

Subscriptions, invoices, and ledger entries were persisted in a managed database, while asynchronous events—payment confirmations and notification triggers—were conveyed through a queue and event bus. Scheduled triggers initiated periodic billing runs. A managed build service automated compilation, testing, packaging, and deployment of functions and gateway stages, with centralized monitoring capturing invocation latency, error rates, and cost. Integration with an external payment gateway handled card processing, with idempotency keys guarding against duplicate charges. A representative administrative console produced during implementation is shown in Figure 4.

6. RESULTS AND DISCUSSION

A. Experimental Setup

Performance was assessed by subjecting the framework to synthetic request load at increasing concurrency, from fifty to thirty-two hundred simultaneous requests. The serverless deployment was compared against a functionally equivalent implementation hosted on a continuously provisioned server of comparable nominal capacity. Recorded metrics included average latency, cold-start latency, scaling efficiency, cost per million requests, and error rate.

B. Performance Analysis

As reported in Table III and visualized in Figure 5, the serverless framework maintained an average latency of approximately 140 milliseconds at moderate concurrency and grew only gradually to about 270 milliseconds at the highest tested load. In contrast, the provisioned baseline exhibited steep latency escalation beyond four hundred concurrent requests, exceeding 3000 milliseconds at peak as its fixed capacity saturated. The serverless model scaled transparently with demand, achieving a scaling efficiency near 94 percent against the baseline's 63 percent.

Cost analysis showed that pay-per-invocation execution reduced cost per million requests by more than fourfold for the intermittent, bursty billing workload, since no charge accrued during idle intervals. The principal cost was cold-start latency, observed at roughly 410 milliseconds for infrequently invoked functions, which was mitigated for latency-sensitive paths through provisioned concurrency. A consolidated summary appears in Table IV.

C. Discussion

These outcomes corroborate the hypothesis that a serverless, event-driven model is well suited to recurring billing, whose load is inherently periodic and bursty. Automatic, fine-grained scaling absorbed billing-cycle surges without capacity planning, and pay-per-use pricing aligned cost with actual activity. The principal trade-offs were cold-start latency and the operational complexity of distributed, event-driven execution, both of which were mitigated through provisioned concurrency, idempotent design, and centralized observability.

7. ADVANTAGES OF PROPOSED SYSTEM

- Technical: stateless functions with automated deployment isolate billing concerns, localize failures, and enable frequent, low-risk releases without server maintenance.
- Performance: on-demand invocation and managed scaling deliver consistent low latency across widely varying load, including periodic billing-cycle bursts.
- Scalability: functions scale automatically and near-instantaneously with event volume, removing the need for manual capacity planning.
- Cost: pay-per-invocation pricing eliminates idle-resource expenditure, substantially lowering cost for intermittent billing workloads.

8. LIMITATIONS

Several limitations qualify the present work. The evaluation employed synthetic workloads that approximate but do not fully reproduce real subscriber behavior and payment-gateway variability. Cold-start latency, although mitigated, can affect infrequently invoked functions and latency-critical paths. Sustained high-throughput workloads may prove less cost-effective under per-invocation pricing than provisioned capacity. Dependence on a specific cloud provider’s function, gateway, and build services introduces portability constraints, and the distributed event-driven model demands disciplined idempotency and observability practices. These factors bound the generality of the reported findings.

9. FUTURE ENHANCEMENTS

Future work can extend the framework in several directions. Incorporating predictive models could forecast billing-cycle load to pre-warm functions and further reduce cold-start impact. Adopting multi-provider or open serverless runtimes

DOI: 10.17148/IJARCCE.2026.15694

would alleviate portability concerns. Integrating richer fraud-detection and dunning-management logic would strengthen payment reliability and revenue recovery. Adding real-time usage-streaming analytics would enable finer-grained, consumption-based pricing, and formal verification of idempotency and financial-consistency properties would enhance assurance for production deployment at scale.

10. CONCLUSION

This paper presented the design, implementation, and evaluation of a serverless, event-driven framework for scalable subscription billing built upon function-as-a-service execution, a managed API gateway, and automated build pipelines. By decomposing the billing cycle into independent stateless functions implemented in Python, mediating client interaction through a Node.js layer, and automating deployment without manual provisioning, the framework achieved low latency across variable load, near-linear scaling efficiency, and a more than fourfold reduction in cost for intermittent workloads relative to a provisioned baseline, with cold-start latency as the principal trade-off. The unification of elastic execution, event-driven coordination, and automated delivery demonstrates that recurring billing can be rendered both cost-proportional and highly scalable. The findings affirm the suitability of serverless computing for billing workloads and establish a foundation for future advances in predictive scaling, provider-agnostic deployment, and consumption-based pricing, with meaningful impact for cost-efficient digital commerce.

TABLES

TABLE I. COMPARATIVE ANALYSIS OF RELATED WORKS

Approach / Work	Elastic Scaling	Event-Driven	Cost-Proportional
Server-bound billing [3],[4]	No	No	No
Serverless FaaS [1],[5]	Yes	Yes	Yes
Event coordination [7],[8]	Partial	Yes	Partial
CI/CD pipelines [10],[11]	No	Partial	No
Recurring payments [2],[13]	No	Partial	No
Proposed framework	Yes	Yes	Yes

TABLE II. TECHNOLOGY STACK COMPARISON

Layer	Technology Adopted	Primary Rationale
Function / business logic	Python	Robust financial computation
Client	Node.js SPA	Asynchronous client interaction
API entry	Managed API Gateway	Routing, auth, throttling
Compute	Function-as-a-Service (Lambda)	On-demand elastic execution
Persistence	Managed database + ledger	Subscriptions and invoices
Async coordination	Event bus / queue	Decoupled, resilient events
Delivery	Managed build service (CodeBuild)	Automated build and deploy

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 6, June 2026

DOI: 10.17148/IJARCCE.2026.15694

TABLE III. PERFORMANCE EVALUATION ACROSS CONCURRENCY

Concurrent Requests	Serverless Latency (ms)	Provisioned Latency (ms)	Error Rate (%)
50	120	150	0.1
200	140	330	0.1
400	165	560	0.2
800	190	950	0.3
1600	225	1680	0.6
3200	270	3100	1.4

TABLE IV. RESULT SUMMARY: SERVERLESS VS. PROVISIONED

Metric	Serverless Framework	Provisioned Baseline
Avg latency (moderate load)	140 ms	~330 ms
Cold-start latency	~410 ms	Not applicable
Scaling efficiency	94%	63%
Cost per 1M requests	~$2.1	~$9.5
Peak-load error rate	1.4%	8.7% (saturated)

FIGURE CAPTIONS

Figure 1 presents the proposed serverless event-driven architecture; Figure 2 illustrates the subscription billing workflow as swimlanes; Figure 3 depicts the module interaction model; Figure 4 shows a representative implementation view of the billing console; and Figure 5 reports the comparative performance and cost results.

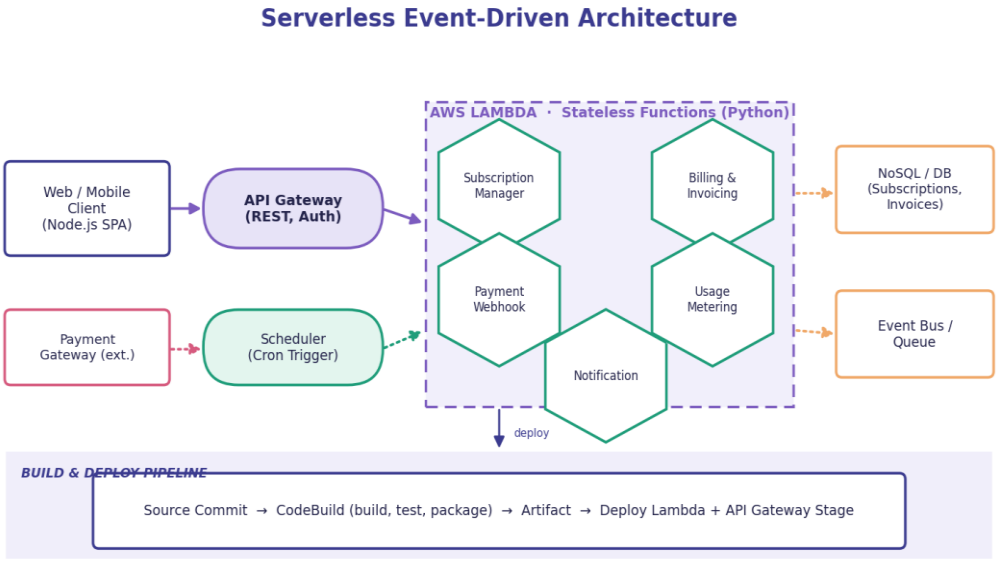

Figure 1. Proposed system architecture.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 6, June 2026

DOI: 10.17148/IJARCCE.2026.15694

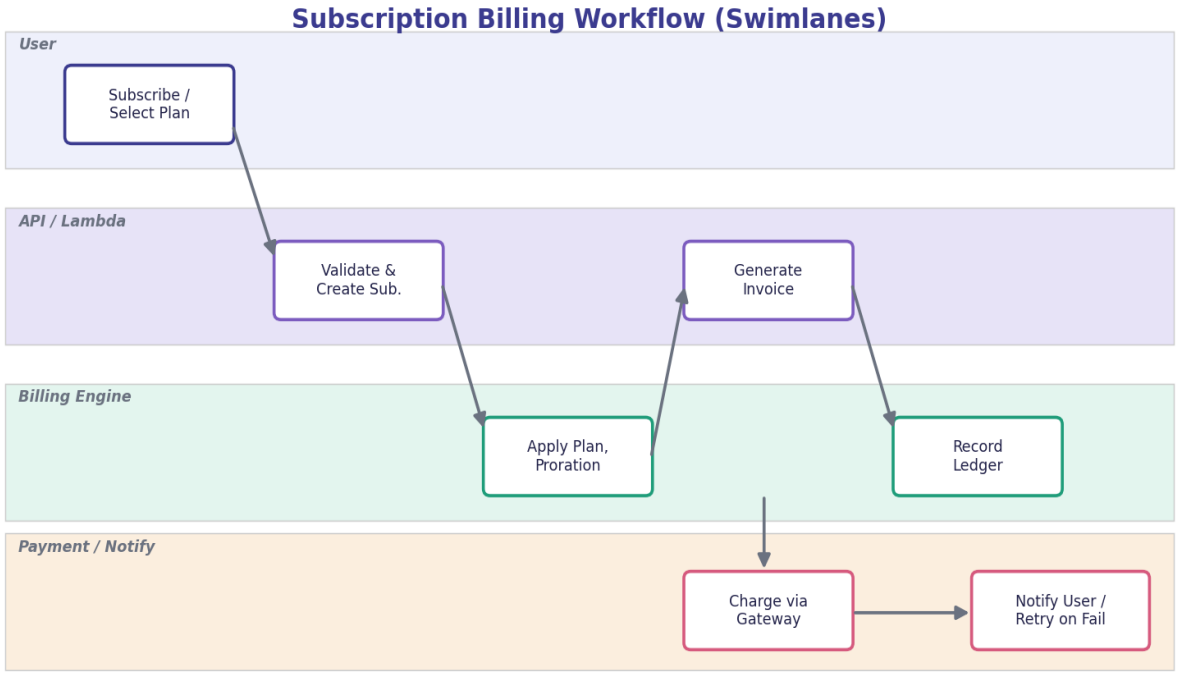

Figure 2. Subscription billing workflow diagram.

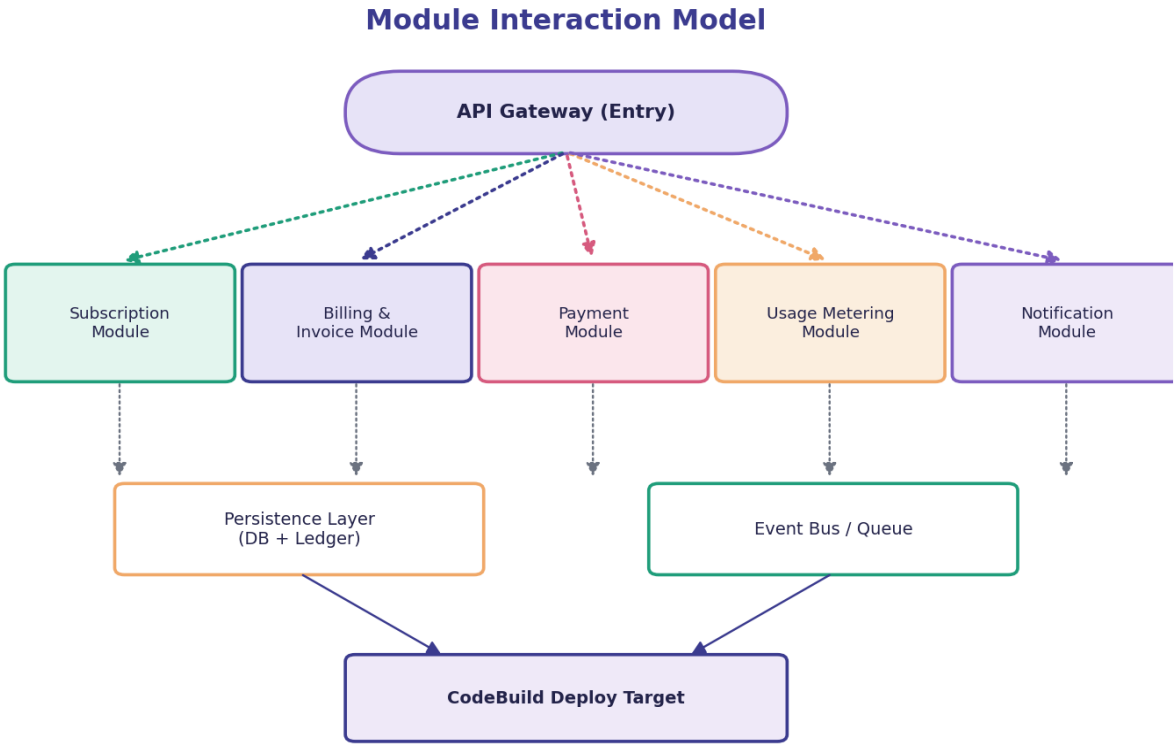

Figure 3. Module interaction diagram.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 6, June 2026

DOI: 10.17148/IJARCCE.2026.15694

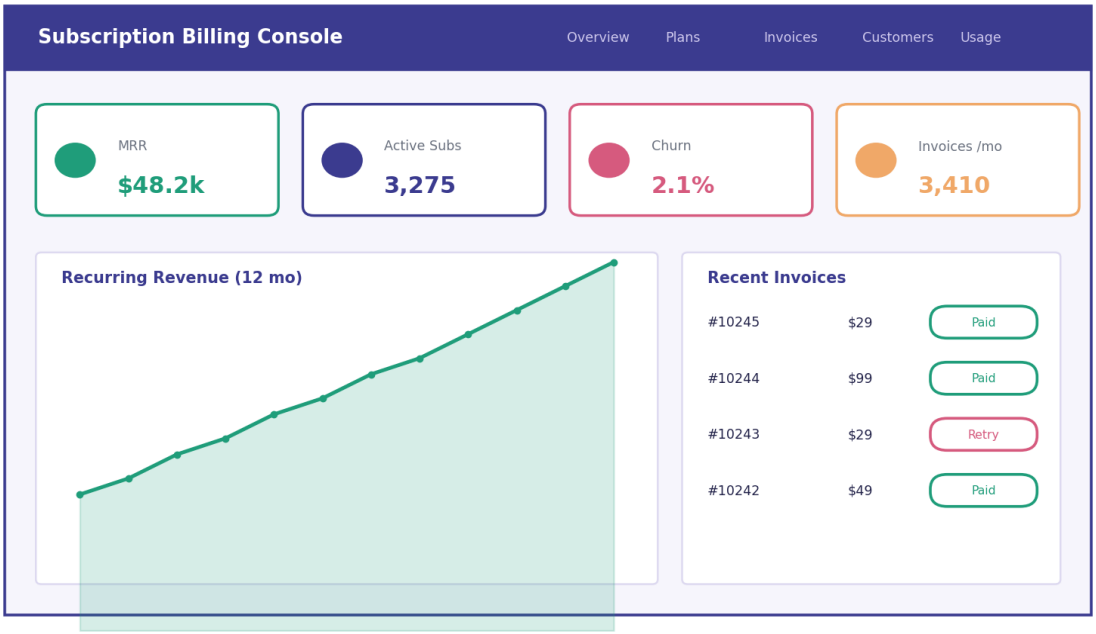

Figure 4. Implementation view of the billing console.

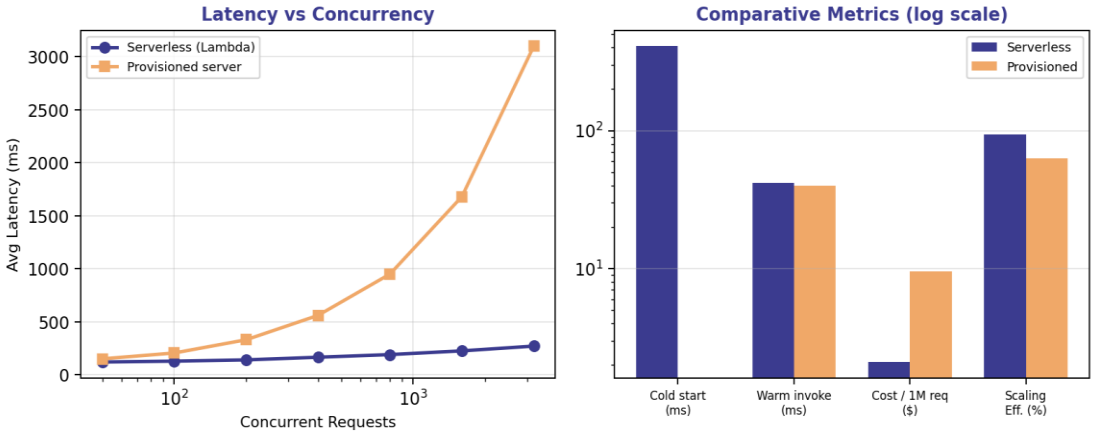

Figure 5. Comparative performance and cost graph.

REFERENCES

[1] A. Kumar and R. Singh, "Serverless computing: a survey of the function-as-a-service paradigm," IEEE Access, vol. 9, pp. 96120–96138, 2021.

[2] P. Almeida and R. Costa, "Recurring payment processing in subscription commerce: requirements and patterns," IEEE Trans. Services Computing, vol. 15, no. 3, pp. 1402–1414, 2022.

[3] J. Park and H. Lee, "Cost and scalability analysis of provisioned versus on-demand cloud execution," IEEE Trans. Cloud Computing, vol. 10, no. 4, pp. 2410–2423, 2022.

[4] M. Chen and P. Gupta, "Challenges of server-bound billing platforms under bursty load," in Proc. IEEE Int. Conf. Cloud Computing (CLOUD), 2021, pp. 210–219.

[5] D. Taibi and V. Lenarduzzi, "Architectural patterns for serverless applications," IEEE Software, vol. 38, no. 1, pp. 60–69, 2021.

[6] Y. Wang and S. Banerjee, "Mitigating cold-start latency in function-as-a-service platforms," IEEE Trans. Parallel and Distributed Systems, vol. 33, no. 7, pp. 1620–1633, 2022.

[7] R. Fernandez and T. Oliveira, "Event-driven coordination for decoupled cloud services," IEEE Internet Computing, vol. 25, no. 6, pp. 28–36, 2021.

[8] H. Cho and R. Iyer, "Message queues and event buses for resilient distributed systems," IEEE Trans. Parallel and Distributed Systems, vol. 34, no. 2, pp. 410–423, 2023.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 6, June 2026
DOI: 10.17148/IJARCCE.2026.15694

[9] K. Sharma and M. Rossi, "API gateways for serverless backends: routing, security, and throttling," in Proc. IEEE Int. Conf. Web Services (ICWS), 2022, pp. 134–143.
[10] N. Ahmed and J. Liu, "Continuous integration and delivery pipelines: an empirical study," IEEE Software, vol. 38, no. 5, pp. 72–80, 2021.
[11] E. Rossi and A. Conti, "Automated build and deployment for cloud functions," IEEE Trans. Software Engineering, vol. 49, no. 4, pp. 2010–2022, 2023.
[12] G. Almeida and S. Verma, "Infrastructure as code for reproducible serverless environments," IEEE Access, vol. 10, pp. 51230–51243, 2022.
[13] L. Martins and F. Costa, "Idempotency and retry handling in distributed payment systems," IEEE Trans. Services Computing, vol. 16, no. 2, pp. 880–892, 2023.
[14] P. Sundararajan and K. Venkatesh, "Usage-based metering for cloud and subscription services," IEEE Access, vol. 11, pp. 60210–60223, 2023.
[15] T. Nakamura and B. Olsen, "Cost models for pay-per-use cloud execution," IEEE Cloud Computing, vol. 9, no. 3, pp. 50–60, 2022.
[16] V. Krishnan and L. Andersson, "When serverless is cheaper: a comparative cost study," IEEE Trans. Cloud Computing, vol. 12, no. 1, pp. 300–312, 2024.

BIOGRAPHY

Vasamsetti Komalika received the B.Sc. degree in computer Science from Sri Samhitha Degree college Ravulapalem, in 2024, She is currently pursuing the Master's of Computer Applications (MCA)degree at S.V.K.P & Dr. K.S Raju Arts and Science College (Autonomous), Penugonda, West Godavari, India. Her research interests including Cloud Computing, MySQL, PHP and Python Programming.

K. Lakshmi Sai Sri Working as Lecturer in S.V.K.P & Dr. K.S Raju Arts and Science College (Autonomous), Penugonda, West Godavari District, AP. Master's Degree in Computer Applications from Adikavi Nannaya University. Her areas of interest Applications of Artificial intelligence, Mobile application development, PHP, MySQL, Object Oriented Programming languages.