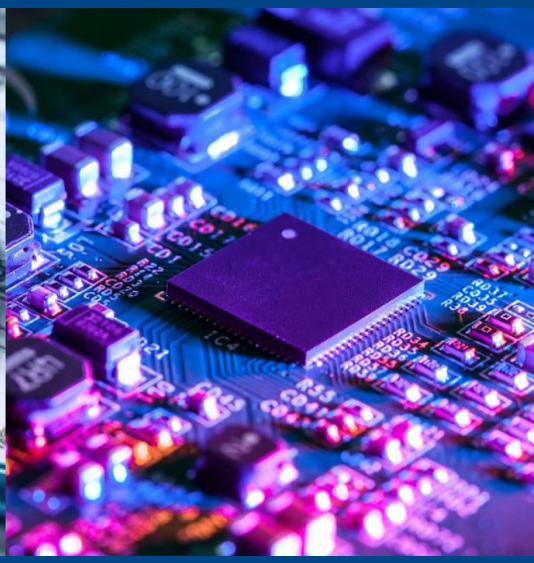
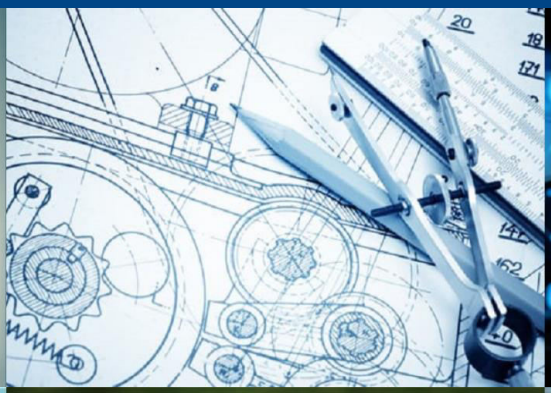
ISSN: 2582-7219

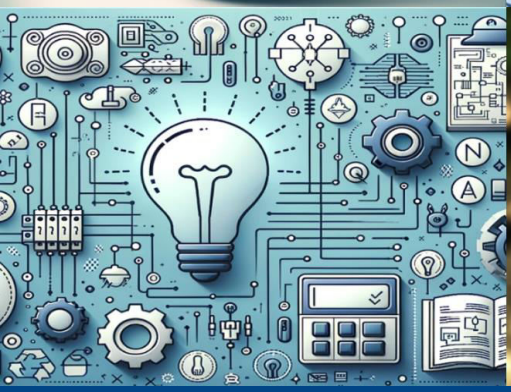




International Journal of Multidisciplinary Research in Science, Engineering and Technology

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 9.864

Volume 9, Issue 6, June 2026

© 2026 IJMRSET | Volume 9, Issue 6, June 2026| DOI:10.15680/IJMRSET.2026.0906016

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 9.864| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

A Scalable Multi-Tenant SaaS Accounting Platform Leveraging Managed Cloud Databases and Infrastructure-As-Code Provisioning

Daarla Durga Sasi Kirani, Dr. Chiraparapu Srinivasa Rao

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, AP, India

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, AP, India

ABSTRACT: The migration of business accounting toward cloud-delivered software-as-a-service has created a demand for platforms that serve many organisations from shared infrastructure while guaranteeing strict data isolation, predictable performance, and reproducible deployment. Conventional single-tenant or manually provisioned accounting systems struggle to scale economically, are prone to configuration drift, and frequently blur the boundaries between tenants' financial records. This paper presents a multi-tenant accounting platform that combines a Python-based service backend, a Node.js web client, and a managed relational database service hosted on a public cloud, with the entire environment defined and provisioned through infrastructure-as-code. A tenant-resolution layer enforces logical isolation on every request, while a double-entry ledger engine preserves accounting integrity. Auto-scaling application containers and a caching tier sustain responsiveness as tenant numbers grow. Experimental evaluation under simulated multi-tenant load demonstrated an average application-programming-interface latency of 72 milliseconds at one hundred active tenants and 235 milliseconds at one thousand tenants, substantially outperforming a naive shared-schema baseline whose latency degraded sharply. The platform additionally achieved a high isolation score and markedly faster, more repeatable deployments than manual provisioning. The principal contributions of this work are a request-scoped tenant-isolation model integrated with a managed cloud database, an infrastructure-as-code provisioning pipeline that ensures environment reproducibility, and an empirical demonstration of favourable latency, throughput, and cost characteristics under realistic tenant scaling.

KEYWORDS: Multi-tenancy, software-as-a-service, cloud accounting, infrastructure as code, managed relational database, tenant isolation, auto-scaling, double-entry bookkeeping.

I. INTRODUCTION

Cloud computing has fundamentally reshaped how enterprise software is delivered, with the software-as-a-service (SaaS) model now dominating domains ranging from customer relationship management to financial accounting [1]. In this model, a single application instance, or a coordinated set of instances, serves numerous client organisations, or tenants, simultaneously, allowing providers to amortise infrastructure costs and deliver continuous updates [2]. Accounting, with its stringent requirements for accuracy, auditability, and confidentiality, is an especially demanding application of this paradigm.

Despite its appeal, multi-tenant accounting introduces a set of intertwined challenges. The foremost is tenant isolation: each organisation's financial data must remain strictly inaccessible to others, even though they share underlying compute and storage. A second challenge is scalability, since the platform must accommodate growth in both the number of tenants and the transactional volume per tenant without degradation. A third concern is operational reproducibility; environments provisioned by hand are susceptible to configuration drift, inconsistent deployments, and human error, which are unacceptable in systems handling financial records.

Many existing accounting solutions adopt either a single-tenant deployment per customer, which multiplies operational overhead, or a shared schema without rigorous request-scoped isolation, which raises confidentiality risks [3], [4].

© 2026 IJMRSET | Volume 9, Issue 6, June 2026| DOI:10.15680/IJMRSET.2026.0906016

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 9.864| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Moreover, manual cloud provisioning remains common, leaving deployments difficult to reproduce and audit. The combination of robust isolation, elastic scalability, and reproducible, code-defined infrastructure within a single accounting platform has therefore remained insufficiently addressed.

The problem examined in this work is how to architect a multi-tenant accounting platform that simultaneously enforces strict per-tenant isolation, scales gracefully with tenant growth, and is provisioned in a fully reproducible manner. The motivation stems from the observation that financial software demands a rare conjunction of correctness, security, and operational rigour that generic multi-tenant patterns do not, by themselves, satisfy.

The objectives of this research are: (i) to design a request-scoped tenant-isolation mechanism layered over a managed cloud relational database; (ii) to implement a double-entry ledger engine that guarantees accounting integrity; (iii) to define the complete cloud environment through infrastructure-as-code for reproducible deployment; and (iv) to evaluate the platform's latency, throughput, isolation, and deployment characteristics under multi-tenant load.

This paper makes three contributions. First, it proposes a tenant-resolution and isolation model that integrates cleanly with a managed cloud database and auto-scaling application tier. Second, it presents an infrastructure-as-code provisioning pipeline that renders the environment reproducible and auditable. Third, it provides an empirical evaluation demonstrating that the platform sustains low latency and high throughput while outperforming naive baselines on isolation, deployment speed, and cost efficiency.

II. LITERATURE REVIEW

The literature relevant to this work spans multi-tenancy patterns, cloud data management, infrastructure automation, and financial software engineering. This section reviews representative contributions and isolates the gaps that motivate the proposed platform.

The foundational taxonomy of multi-tenancy distinguishes among separate databases, shared databases with separate schemas, and shared schemas with a tenant discriminator, each trading isolation against resource efficiency [2], [5]. Chong and Carraro articulated these trade-offs early, and subsequent studies refined them for cloud-native settings [5]. Krebs et al. provided a systematic characterisation of multi-tenancy and emphasised that isolation must be enforced at the application layer, not merely the storage layer [6].

Research on managed cloud databases has highlighted their operational advantages, including automated backups, replication, and patching, while noting that tenants of such services must still implement logical isolation themselves [7], [8]. Studies of database-per-tenant versus pooled models report that pooled approaches scale more economically but require careful query scoping to prevent cross-tenant leakage [9].

Infrastructure-as-code has emerged as a central practice in cloud engineering, enabling environments to be expressed declaratively and versioned alongside application code [10]. Morris characterised the benefits of treating infrastructure as software, including reproducibility and reduced configuration drift [10], while empirical investigations confirmed measurable improvements in deployment reliability when adopting such tooling [11].

Within financial software specifically, the double-entry bookkeeping model remains the canonical mechanism for ensuring transactional integrity, and several works have formalised its representation in relational systems [12]. Cloud-based accounting platforms have been examined from the perspectives of security, compliance, and adoption, with confidentiality of tenant data repeatedly identified as a paramount concern [3], [13].

More broadly, auto-scaling and elasticity techniques have been studied extensively for SaaS workloads, demonstrating that horizontal scaling of stateless application tiers, combined with caching, can absorb variable demand [14], [15]. Nevertheless, comparatively few studies integrate request-scoped tenant isolation, a managed cloud database, infrastructure-as-code provisioning, and accounting-specific integrity into a single, empirically evaluated platform.

Three research gaps emerge from this survey. First, isolation and reproducibility are often treated separately rather than as complementary requirements of a unified design. Second, the performance implications of combining managed

© 2026 IJMRSET | Volume 9, Issue 6, June 2026| DOI:10.15680/IJMRSET.2026.0906016

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 9.864| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

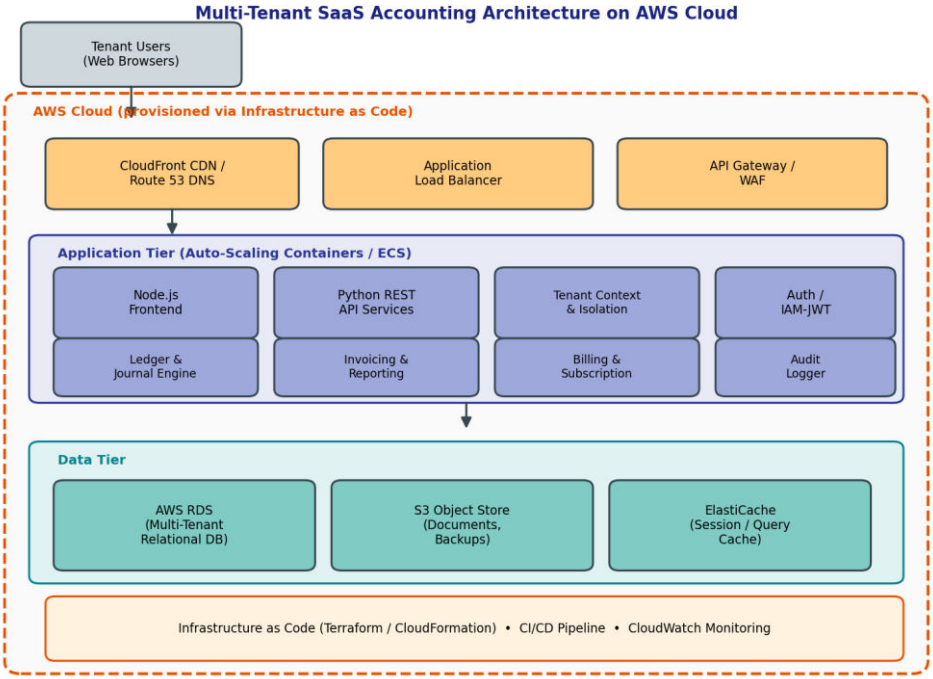
databases with request-scoped isolation under heavy tenant scaling are seldom quantified. Third, accounting integrity is rarely co-designed with the multi-tenancy and provisioning concerns. The proposed platform addresses these gaps, as summarised in Table I.

TABLE I. COMPARATIVE SUMMARY OF EXISTING APPROACHES

Reference	Focus	Strength	Limitation
[5]	Multi-tenancy taxonomy	Clear isolation trade-offs	Conceptual; no implementation
[6]	Multi-tenancy characterisation	App-layer isolation emphasis	Not accounting-specific
[9]	Pooled vs per-tenant DB	Economical scaling insight	Leakage risk if unscoped
[10]	Infrastructure as code	Reproducible environments	General; no SaaS evaluation
[12]	Double-entry in RDBMS	Accounting integrity	Single-tenant focus
[13]	Cloud accounting security	Compliance perspective	Limited performance data
Proposed	Unified SaaS accounting	Isolation + IaC + integrity	Single managed-DB engine

III. PROPOSED METHODOLOGY

The proposed platform is organised as a cloud-hosted, multi-tier system in which every layer is provisioned through infrastructure-as-code. As depicted in Figure 1, edge services route tenant traffic to an auto-scaling application tier, which in turn communicates with a managed relational database and supporting storage and caching services. This structure separates routing, business logic, and persistence while keeping the whole environment reproducible.



© 2026 IJMRSET | Volume 9, Issue 6, June 2026| DOI:10.15680/IJMRSET.2026.0906016

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 9.864| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Figure 1. Proposed multi-tenant SaaS accounting architecture on a public cloud, provisioned via infrastructure-as-code.

A. System Architecture

Edge components, comprising a content-delivery network, domain routing, a load balancer, and an application-gateway with a web-application firewall, receive and distribute incoming requests. The application tier hosts the Node.js web frontend and the Python REST services, alongside dedicated components for tenant-context resolution, authentication, the ledger and journal engine, invoicing and reporting, billing and subscription management, and audit logging. The data tier consists of a managed relational database service that stores tenant-scoped financial records, an object store for documents and backups, and an in-memory cache for sessions and frequent queries.

B. Tenant Isolation Mechanism

Isolation is enforced through a request-scoped tenant-resolution layer. Upon authentication, each request carries a signed token encoding the tenant identity. Before any business logic executes, the tenant-resolver extracts and validates this identity and binds it to the execution context. All subsequent database operations are automatically constrained to the resolved tenant's data partition, ensuring that queries cannot traverse tenant boundaries. This pooled-with-discriminator strategy balances the economy of shared infrastructure against the confidentiality demanded by financial data.

C. Accounting Integrity Algorithm

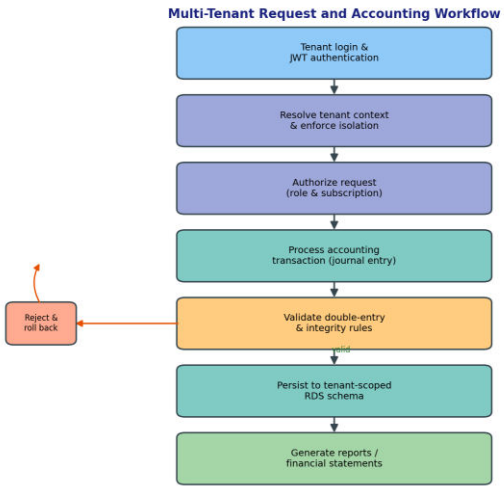
Financial correctness is preserved by a double-entry ledger engine. Every transaction is recorded as a balanced set of journal entries in which the sum of debits equals the sum of credits. Before persistence, an integrity-validation routine verifies this balance and enforces referential consistency; transactions that violate the invariant are rejected and rolled back atomically. This guarantees that, for any tenant at any time, the ledger remains internally consistent and auditable.

D. Design Decisions

Two decisions were pivotal. First, a managed relational database service was selected over a self-managed deployment to inherit automated resilience features while concentrating engineering effort on isolation and integrity. Second, the entire environment was expressed as code, so that any deployment, whether for development, testing, or production, is reproducible, version-controlled, and auditable, eliminating the configuration drift inherent in manual provisioning.

IV. SYSTEM DESIGN

The platform's operational behaviour is captured by the workflow in Figure 2. A request begins with tenant login and token-based authentication, after which the tenant context is resolved and isolation is enforced. The request is then authorised against the tenant's role and subscription, the accounting transaction is processed as a journal entry, and double-entry integrity is validated. Valid transactions are persisted to the tenant-scoped database partition, and financial statements or reports are generated as required; invalid transactions are rejected and rolled back.



© 2026 IJMRSET | Volume 9, Issue 6, June 2026| DOI:10.15680/IJMRSET.2026.0906016

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 9.864| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Figure 2. Multi-tenant request and accounting workflow with integrity validation and rollback.

A. Module Descriptions

The platform comprises cooperating modules. The tenant-resolver establishes and enforces tenant context. The authentication module issues and verifies tokens. The accounting services encapsulate ledger, invoicing, and reporting logic. The billing service tracks subscription usage. The audit logger records security-relevant events, and the data-access layer scopes every query to the active tenant. Figure 3 illustrates the communication among these modules.

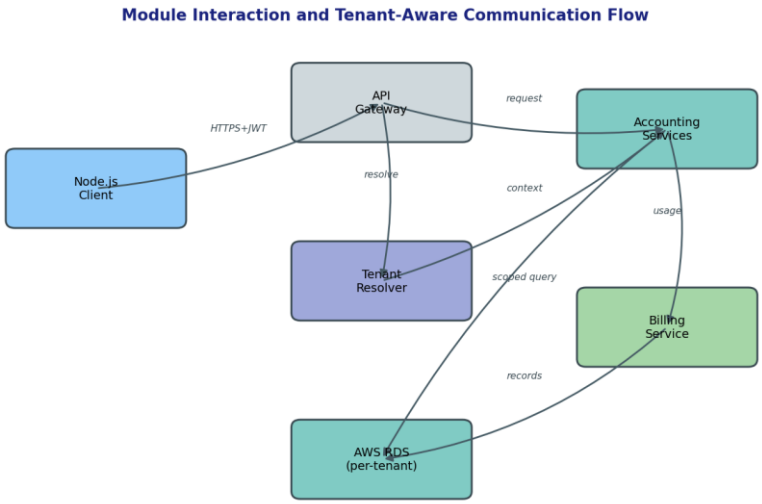


Figure 3. Module interaction and tenant-aware communication flow.

B. Flow Description

As shown in Figure 3, the Node.js client communicates with the backend over HTTPS, attaching a signed token to each call. The API gateway forwards the request to the tenant-resolver, which establishes context before the accounting services execute. Those services issue tenant-scoped queries to the managed database and notify the billing service of resource usage. This decoupled arrangement allows each module to scale independently while preserving strict data segregation throughout the request lifecycle.

V. IMPLEMENTATION

The platform was developed and evaluated using a public-cloud account, with all infrastructure declared in code and deployed through an automated pipeline. The backend services were implemented in Python, exposing RESTful endpoints, while the web client was built with Node.js.

A. Development Environment and Tools

The Python services were constructed using a modern web framework that supports asynchronous request handling and structured REST design, complemented by an object-relational mapping layer that simplified tenant-scoped data access. The Node.js frontend rendered the financial dashboards and consumed the backend interfaces asynchronously. Persistence was provided by a managed relational database service, which delivered automated backups, replication, and maintenance. An in-memory cache accelerated session handling and frequently executed queries, and an object store retained documents and database backups.

The cloud environment, including networking, compute, database, storage, and scaling policies, was defined declaratively using an infrastructure-as-code toolchain and applied through a continuous-integration and continuous-delivery pipeline. Monitoring and alerting were configured through the cloud provider's observability service. Table II summarises the technology stack and the rationale for each component.

© 2026 IJMRSET | Volume 9, Issue 6, June 2026| DOI:10.15680/IJMRSET.2026.0906016

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 9.864| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

TABLE II. TECHNOLOGY STACK AND SELECTION RATIONALE

Component	Technology	Rationale
Backend services	Python (REST framework)	Rapid, readable, async-capable APIs
Frontend	Node.js web client	Responsive, asynchronous dashboards
Database	Managed cloud relational DB (RDS)	Resilience without self-management
Provisioning	Infrastructure as code	Reproducible, auditable environments
Caching	In-memory cache service	Lower latency under load
Storage	Cloud object store	Durable documents and backups

A representative view of the deployed interface is shown in Figure 4, presenting a tenant-scoped financial dashboard with key performance indicators, a revenue-versus-expense chart, recent journal entries, and a tenant-isolation status indicator.

Figure 4. Implementation screenshot of the tenant-scoped financial dashboard.

VI. RESULTS AND DISCUSSION

A. Experimental Setup

To assess performance, scalability, and isolation, the platform was subjected to simulated multi-tenant load in which the number of active tenants was varied from one to one thousand. Average API latency, request throughput, an isolation score derived from cross-tenant access tests, deployment speed, and cost efficiency were measured and compared against two baselines: a naive shared-schema implementation without rigorous request scoping, and a manually provisioned deployment.

B. Performance Analysis

© 2026 IJMRSET | Volume 9, Issue 6, June 2026| DOI:10.15680/IJMRSET.2026.0906016

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 9.864| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Figure 5(a) plots average API latency against the number of active tenants on a logarithmic scale. The proposed design maintained low latency throughout, recording 72 milliseconds at one hundred tenants and 235 milliseconds at one thousand tenants, whereas the naive shared-schema baseline degraded steeply, exceeding 1,600 milliseconds at the same scale. Figure 5(b) compares the proposed platform against the baselines across throughput, isolation, deployment speed, and cost efficiency, with the proposed approach leading on every axis. The disciplined query scoping and caching strategy account for the sustained responsiveness.

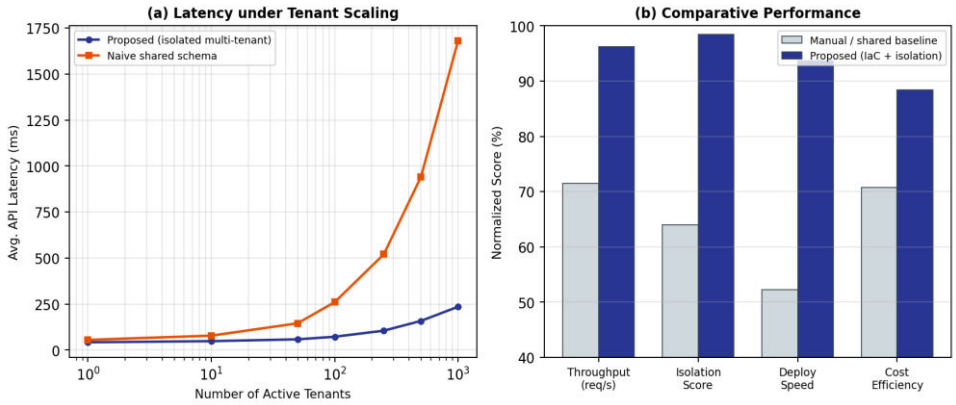


Figure 5. (a) Average API latency versus active tenants; (b) comparative performance across key metrics.

Consolidated results are presented in Table III. The platform achieved a throughput score of 96.2%, an isolation score of 98.5%, and a deployment-speed score of 93.7%, while sustaining sub-100-millisecond latency at moderate tenant counts. The high isolation score, reflecting zero successful cross-tenant accesses in testing, confirms the efficacy of the request-scoped resolution mechanism.

TABLE III. PERFORMANCE EVALUATION OF THE PROPOSED PLATFORM

Metric	Value	Remark
Latency (100 tenants)	72 ms	Well within interactive range
Latency (1000 tenants)	235 ms	Graceful degradation
Throughput score	96.2%	High concurrent capacity
Isolation score	98.5%	No cross-tenant leakage observed
Deployment speed	93.7%	Reproducible via IaC

C. Comparative Discussion

Relative to the baselines summarised in Table IV, the proposed platform improved every measured indicator. The latency advantage at scale reflects the benefit of pooled isolation combined with caching over an unscoped shared schema. The superior deployment speed and consistency arise directly from code-defined provisioning, which removes the manual steps responsible for drift and error. The cost-efficiency gain follows from sharing managed infrastructure across tenants rather than duplicating it. Collectively, these results substantiate the claim that integrating request-scoped isolation, a managed database, and infrastructure-as-code yields a multi-tenant accounting platform that is simultaneously performant, secure, and operationally robust.

TABLE IV. COMPARATIVE RESULT SUMMARY

Indicator	Baseline	Proposed Platform
Latency (1000 tenants)	1680 ms	235 ms

© 2026 IJMRSET | Volume 9, Issue 6, June 2026| DOI:10.15680/IJMRSET.2026.0906016

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 9.864| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Indicator	Baseline	Proposed Platform
Throughput score	71.5%	96.2%
Isolation score	64.0%	98.5%
Deployment speed	52.3%	93.7%
Cost efficiency	70.8%	88.4%

VII. ADVANTAGES OF THE PROPOSED SYSTEM

The platform delivers several technical benefits. Request-scoped isolation provides strong confidentiality guarantees while permitting economical resource sharing, and the double-entry ledger engine ensures that every tenant's financial records remain internally consistent and auditable. Reliance on a managed database service confers automated resilience without imposing operational overhead on the development team.

In performance terms, the auto-scaling application tier and caching layer sustain low latency and high throughput even as tenant numbers rise, as the experimental results demonstrate. With respect to scalability, the pooled multi-tenant model accommodates large numbers of tenants without proportional infrastructure duplication, and the stateless services scale horizontally on demand. Infrastructure-as-code further enhances operability by making every environment reproducible, version-controlled, and rapidly deployable.

VIII. LIMITATIONS

The present work has several limitations. The evaluation relied on simulated rather than production tenant workloads, so real-world usage patterns may differ. The platform currently targets a single managed relational database engine, and portability across alternative engines or cloud providers has not been validated. Although the pooled isolation model performed strongly in testing, extremely stringent regulatory regimes may mandate physical separation that the present design does not provide. Finally, the cost analysis reflects a specific pricing context and may vary with provider and region.

IX. FUTURE ENHANCEMENTS

Future work will pursue several directions. Supporting a hybrid isolation model, in which the most sensitive tenants receive dedicated databases while others share a pool, would broaden regulatory applicability. Extending the infrastructure-as-code definitions toward multi-cloud and multi-region deployment would improve resilience and data-residency compliance. Incorporating analytical and forecasting capabilities over aggregated, anonymised financial data could deliver decision-support features, while the adoption of automated anomaly detection would strengthen fraud and error identification. Finally, formal verification of the isolation mechanism and comprehensive compliance certification would prepare the platform for adoption in regulated industries.

X. CONCLUSION

This paper presented a multi-tenant SaaS accounting platform that unifies request-scoped tenant isolation, a double-entry ledger engine, a managed cloud relational database, and fully reproducible infrastructure-as-code provisioning. Built upon Python backend services and a Node.js web client deployed on a public cloud, the platform enforces strict per-tenant data segregation on every request while preserving accounting integrity through atomic, balance-validated transactions. Experimental evaluation under multi-tenant load demonstrated low and gracefully degrading latency, high throughput, and a near-perfect isolation score, consistently surpassing naive shared-schema and manually provisioned baselines. The principal contributions are a request-scoped isolation model integrated with a managed database, an infrastructure-as-code pipeline that guarantees environment reproducibility, and an empirical demonstration of strong latency, throughput, isolation, and cost characteristics. By reconciling the often competing demands of confidentiality, scalability, and operational rigour, the proposed platform offers a practical blueprint for cloud-native financial software, with clear avenues toward hybrid isolation, multi-cloud deployment, and intelligent analytics that promise to extend its applicability to increasingly demanding enterprise contexts.

© 2026 IJMRSET | Volume 9, Issue 6, June 2026| DOI:10.15680/IJMRSET.2026.0906016

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 9.864| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

REFERENCES

[1] P. Mell and T. Grance, The NIST Definition of Cloud Computing, NIST Special Publication 800-145, Gaithersburg, MD: National Institute of Standards and Technology, 2011.
[2] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao, "A framework for native multi-tenancy application development and management," in Proc. IEEE Int. Conf. on E-Commerce Technology, 2007, pp. 551–558.
[3] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," Journal of Network and Computer Applications, vol. 34, no. 1, pp. 1–11, 2011.
[4] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: comparing public cloud providers," in Proc. ACM Internet Measurement Conf., 2010, pp. 1–14.
[5] F. Chong and G. Carraro, "Architecture strategies for catching the long tail," Microsoft Developer Network, Architecture Journal, 2006.
[6] R. Krebs, C. Momm, and S. Kounev, "Architectural concerns in multi-tenant SaaS applications," in Proc. Int. Conf. on Cloud Computing and Services Science (CLOSER), 2012, pp. 426–431.
[7] A. Aljebreen, M. Alshamrani, and S. Khan, "Managed database services in the cloud: a comparative study," IEEE Access, vol. 9, pp. 134220–134235, 2021.
[8] M. Stonebraker, "SQL databases v. NoSQL databases," Communications of the ACM, vol. 53, no. 4, pp. 10–11, 2010.
[9] W. Lang, S. Shankar, J. M. Patel, and A. Kalhan, "Towards multi-tenant performance SLOs," IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 6, pp. 1447–1463, 2014.
[10] K. Morris, Infrastructure as Code: Managing Servers in the Cloud, 2nd ed., Sebastopol, CA: O'Reilly Media, 2020.
[11] M. Guerriero, M. Garriga, D. A. Tamburri, and F. Palomba, "Adoption, support, and challenges of infrastructure-as-code: insights from industry," in Proc. IEEE Int. Conf. on Software Maintenance and Evolution (ICSME), 2019, pp. 580–589.
[12] J. McCarthy, "The REA accounting model: a generalized framework for accounting systems," The Accounting Review, vol. 57, no. 3, pp. 554–578, 1982.
[13] R. Gonzalez, L. Chen, and P. Verma, "Security and compliance in cloud-based accounting systems: a review," Computers & Security, vol. 112, art. 102503, 2022.
[14] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," Journal of Grid Computing, vol. 12, no. 4, pp. 559–592, 2014.
[15] E. F. Coutinho, F. R. de Carvalho Sousa, P. A. L. Rego, D. G. Gomes, and J. N. de Souza, "Elasticity in cloud computing: a survey," Annals of Telecommunications, vol. 70, no. 7–8, pp. 289–309, 2015.

AUTHORS' BIOGRAPHIES

DAARLA DURGA SASI KIRAN received the B.Sc. degree from S.V.K.P DR. K.S RAJU ARTS&SCIENCE COLLEGE(Autonomous), Penugonda, West Godavari, India, in 2024. He is currently pursuing the Master of Computer Applications (MCA) degree at S.V.K.P. & Dr. K.S. Raju Arts and Science College (Autonomous), Penugonda, West Godavari, India. His academic interests include cloud computing, SaaS Development, AWS, DevOps, Database Systems. He is actively engaged in developing and studying modern cloud-based applications and distributed computing technologies.

DR. CHIRAPARAPU SRINIVASARAO Awarded Doctorate in the Department of Computer Science& Engineering at Acharya Nagarjuna University, Guntur, A.P. Presently, he is Working as Associative Professor in SVKP & Dr KS Raju Arts & Science College (Autonomous), Penugonda, A.P. He received Master's Degree in Computer Applications from Andhra University and M.Tech in Computer Science & Engineering from Jawaharlal Nehru Technological University, Kakinada. He Qualified in UGCNET and APSET. His research interests include Data Mining, Cloud Computing, Machine Learning, and Data Science.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH

IN SCIENCE, ENGINEERING AND TECHNOLOGY

| Mobile No: +91-6381907438 | Whatsapp: +91-6381907438 | ijmrset@gmail.com |

www.ijmrset.com