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.2026.15662

Intelligent Serverless Architecture for Real-Time Expense Management: A Function-as-a-Service Approach to Scalable Financial Analytics

Gudelli Mounika[1], A.N.Rama Mani[2]

MCA Student, Adikavi Nannaya University, SVKP & Dr. K.S. Raju Arts and Science College, Penugonda[1]

Associate Professor, Department Of Master of Computer Applications, Adikavi Nannaya University, SVKP & Dr. K.S. Raju Arts and Science College, Penugonda[2]

Abstract: Effective management of organizational expenditure in distributed computing environments presents persistent challenges, particularly with respect to cost attribution, real-time visibility, and scalable analytics. This paper presents an intelligent, serverless expense management framework built upon Function-as-a-Service (FaaS) paradigms, enabling automated, event-driven financial tracking across multi-tenant cloud deployments. The proposed system integrates a micro services-based backend leveraging AWS Lambda, Azure Functions, or equivalent serverless runtimes with a dynamic, responsive frontend, forming a cohesive full-stack platform for expenditure capture, categorization, and reporting. The architecture incorporates role-based access control (RBAC), multi-dimensional cost aggregation, predictive budget forecasting using lightweight regression models, and RESTful API orchestration through a managed API gateway. Experimental evaluations demonstrate that the proposed framework achieves average API response latencies below 220 ms under concurrent load, reduces infrastructure operational overhead by approximately 63% compared to traditional server-based deployments, and attains a budget-forecasting accuracy of 91.4% measured by mean absolute percentage error (MAPE). The system further supports automated alert generation upon threshold breaches and provides drill-down analytical dashboards for stakeholders. Comparative benchmarking against conventional monolithic and containerized expense systems confirms the superiority of the serverless paradigm in elasticity, cost efficiency, and developer agility. The proposed methodology provides a replicable blueprint for institutions seeking to modernize their financial governance infrastructure through cloud-native technologies.

Keywords: Serverless Computing; Function-as-a-Service; Expense Management; Cloud-Native Architecture; Financial Analytics; Budget Forecasting; Micro services; REST API

I. INTRODUCTION

A. Background

The proliferation of cloud computing has fundamentally transformed how organizations provision, scale, and govern digital resources. Within enterprise ecosystems, expenditure tracking and financial accountability are critical operational requirements, yet legacy expense management systems are frequently constrained by monolithic architectures that impose rigid scaling ceilings and high maintenance burdens [1]. The evolution toward cloud-native paradigms, particularly serverless computing, introduces a compelling alternative that decouples compute capacity from infrastructure management, enabling pay-per-execution billing models that align infrastructure costs directly with actual usage [2].

Serverless architectures, anchored by Function-as-a-Service platforms, abstract server provisioning entirely, automatically scaling from zero to peak demand and returning to idle states without manual intervention [3]. When applied to financial management systems, this paradigm offers intrinsic elasticity suited to the bursty, event-driven nature of expense submission, approval workflows, and analytical query processing. Despite this promise, the scholarly literature reveals a relative scarcity of domain-specific serverless implementations tailored to organizational expense lifecycle management.

B. Problem Statement

Contemporary expense management platforms suffer from several well-documented deficiencies. First, traditional server-resident deployments incur persistent infrastructure costs irrespective of actual utilization, leading to significant

 This work is licensed under a Creative Commons Attribution 4.0 International License

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.2026.15662

resource waste during off-peak periods [4]. Second, the absence of real-time analytics capabilities prevents financial officers from obtaining timely insights necessary for proactive budget governance. Third, monolithic codebases impede the rapid integration of additional financial modules, such as multi-currency support, tax compliance engines, or predictive analytics components [5]. Finally, the operational complexity associated with maintaining containerized or virtual machine-based backends elevates the total cost of ownership and introduces deployment bottlenecks that hinder organizational agility.

C. Research Objectives

This study pursues the following primary objectives: (i) to design and implement a serverless, event-driven expense management system that eliminates persistent infrastructure overhead; (ii) to develop lightweight predictive budget forecasting capabilities deployable within FaaS execution contexts; (iii) to evaluate system performance across dimensions of latency, throughput, cost efficiency, and forecast accuracy; and (iv) to compare the proposed architecture against conventional monolithic and containerized implementations to establish its relative performance advantages.

D. Contributions

The principal contributions of this research are as follows: (1) A novel full-stack serverless expense management architecture integrating event-driven functions, a managed API gateway, NoSQL persistence, and a reactive analytical frontend. (2) A RBAC-enabled multi-tenant data model supporting organizational hierarchies with user-, department-, and enterprise-level cost aggregation. (3) An adaptive budget forecasting module employing exponential smoothing and polynomial regression within cold-start-optimized serverless functions. (4) Empirical benchmarking results establishing performance benchmarks for serverless financial management systems, along with an analysis of cold-start mitigation strategies. (5) A reproducible open-source reference implementation demonstrating the practical viability of the proposed approach.

II. LITERATURE REVIEW

A. Related Work

The academic exploration of serverless computing architectures has expanded substantially since the popularization of AWS Lambda in 2014. Baldini et al. [6] provided one of the earliest systematic analyses of serverless computing models, identifying key programming constraints including statelessness requirements, execution time limits, and cold-start latency as fundamental performance considerations. Their work established the foundational vocabulary and evaluation criteria that subsequent researchers have adopted and refined.

In the domain of financial systems, Wang et al. [7] investigated microservices decomposition strategies for banking applications, demonstrating that modular architectures yield measurable improvements in fault isolation and deployment frequency. However, their implementation retained traditional always-on server deployments rather than embracing FaaS execution models, leaving serverless financial applications as an underexplored territory. Kumari and Pillai [8] addressed expense reporting systems in enterprise resource planning (ERP) contexts, proposing rule-based automation for receipt classification, but the study was conducted within monolithic Java EE environments without consideration of elastic scaling requirements.

Jonas et al. [9] conducted a comprehensive survey of serverless computing challenges, cataloguing issues including vendor lock-in, limited local debugging tooling, and the statistical unpredictability of cold-start latencies across diverse provider implementations. Their recommendation for event-driven architectural patterns aligned with queue-based decoupling informed the present work's asynchronous processing design. Similarly, Eismann et al. [10] developed a taxonomy of serverless use cases, categorizing FaaS workloads by access pattern, statefulness requirement, and latency sensitivity, a framework directly applicable to financial transaction processing scenarios.

Regarding cost optimization in cloud environments, Xu et al. [11] proposed dynamic workload scheduling algorithms for heterogeneous cloud clusters, achieving cost reductions of up to 38% compared to static provisioning approaches. While their work targeted general-purpose compute workloads, the underlying optimization principles are transferable to serverless financial systems where function invocation frequency and execution duration are primary cost determinants. More recently, Nupponen and Taibi [12] reviewed fifteen empirical studies of serverless adoption, identifying scalability, reduced operational complexity, and faster time-to-market as consistently reported benefits, while noting that monitoring observability and inter-function communication overhead represent persistent engineering challenges.

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.2026.15662

B. Research Gap Analysis

A critical examination of the reviewed literature reveals three principal gaps. First, no prior study has presented an end-to-end serverless architecture specifically engineered for the full expense management lifecycle, encompassing submission, multi-level approval workflows, categorization, analytics, and predictive forecasting within a unified FaaS framework. Second, existing financial analytics implementations predominantly rely on periodically refreshed batch computations rather than real-time event-driven aggregation, limiting the currency of reporting insights. Third, the empirical cold-start mitigation strategies documented in general serverless literature have not been evaluated within the specific execution profile of financial application functions, which typically exhibit irregular invocation patterns concentrated within business hours. The present research directly addresses these identified gaps.

C. Comparative Study of Existing Approaches

TABLE I
Comparative Analysis of Existing Expense Management Architectures

Reference	Architecture	Scalability	Real-Time Analytics	Cost Model	Limitations
Wang et al. [7]	Microservices + VMs	High	No	Reserved instances	Always-on cost; no FaaS
Kumari & Pillai [8]	Monolithic ERP	Low	No	On-premises	Rigid scaling; high TCO
Jonas et al. [9]	Serverless Survey	High	Partial	Pay-per-use	Cold-start unaddressed
Xu et al. [11]	Dynamic Scheduling	Medium	No	Spot + reserved	General workloads only
Eismann et al. [10]	FaaS Taxonomy	High	Partial	Pay-per-invocation	No financial domain
Proposed System	**Serverless FaaS**	**Elastic/Auto**	**Yes (Real-Time)**	**Pay-per-execution**	**Cold-start mitigated**

III. PROPOSED METHODOLOGY

A. System Architecture

The proposed system adopts a layered, cloud-native architecture composed of four principal tiers: (1) a Presentation Layer comprising a single-page application (SPA) built with a modern JavaScript framework providing reactive, component-driven user interfaces; (2) an API Gateway Layer responsible for request routing, authentication token validation, and rate limiting; (3) a Serverless Compute Layer housing independently deployable functions organized by business domain; and (4) a Persistence Layer integrating a NoSQL document store for transactional records and a time-series data structure for analytical aggregations.

Figure 1 illustrates the high-level system architecture. Expense submission events originate at the client layer and traverse a managed API gateway that enforces JSON Web Token (JWT) authentication before routing requests to the appropriate FaaS handler. Approved expenses trigger downstream analytics aggregation functions asynchronously via a message queue, decoupling the transactional write path from the analytical read path to ensure consistent sub-300 ms user-perceived response times.

B. Workflow Description

The operational workflow of the proposed system encompasses six sequential phases. In Phase 1, an authenticated user submits an expense record via the client interface, supplying metadata including amount, currency, category, project code, and supporting receipt attachments. In Phase 2, the API gateway validates the request against the user's JWT claims and forwards the payload to the Expense Ingestion Function. In Phase 3, this function performs schema validation, currency normalization to a configurable base currency using live exchange rate lookups, and persists the record to the document store with a status of "Pending Approval."

 This work is licensed under a Creative Commons Attribution 4.0 International License

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.2026.15662

In Phase 4, an approval notification event is published to a message queue, triggering the Notification Dispatch Function which delivers email or in-application alerts to the designated approver(s) based on organizational hierarchy rules. In Phase 5, upon managerial approval or rejection, a status-update event triggers the Analytics Aggregation Function, which atomically updates departmental and project-level spending summaries in the time-series data structure. In Phase 6, the Dashboard Query Function serves pre-aggregated analytics to the frontend, supporting drill-down navigation from enterprise to individual expense records with sub-200 ms response latency.

C. Algorithms and Models

The system employs two complementary computational models for financial intelligence.

Budget Forecasting via Exponential Smoothing: The short-horizon budget consumption forecast is computed using Holt's double exponential smoothing, which accounts for both level and trend components in historical expenditure time-series data. The smoothed level ℓ_t and trend β_t at discrete time step t are computed iteratively as follows:

$$\ell_t = \alpha \cdot y_t + (1 - \alpha)(\ell_{t-1} + \beta_{t-1}) \quad (1)$$

$$\beta_t = \gamma \cdot (\ell_t - \ell_{t-1}) + (1 - \gamma)\, \beta_{t-1} \quad (2)$$

where y_t denotes the observed cumulative expenditure at period t, $\alpha \in (0,1)$ is the level smoothing coefficient, and $\gamma \in (0,1)$ is the trend smoothing coefficient. The h-step-ahead forecast $\hat{y}_{t+h}$ is then given by:

$$\hat{y}_{t+h} = \ell_t + h \cdot \beta_t \quad (3)$$

Optimal values for α and γ are determined through grid search minimizing the sum of squared forecast errors on a rolling validation window of the preceding 90-day historical record for each budget period.

Anomaly Detection via Z-Score Thresholding: Individual expense records are subjected to statistical anomaly screening using a rolling Z-score computed over category-specific expenditure distributions. For an expense amount x submitted under category c, the anomaly score Z_c is defined as:

$$Z_c = (x - \mu_c) / \sigma_c \quad (4)$$

where μ_c and σ_c represent the rolling mean and standard deviation of the most recent 180 approved expense records within category c. Records with $|Z_c| > 2.5$ are flagged for enhanced approval scrutiny and logged for audit trail purposes.

D. Implementation Details

The serverless functions are implemented in Node.js 18 LTS, chosen for its low cold-start overhead relative to JVM-based runtimes in the target FaaS environment. Each function adheres to a single-responsibility principle, handling precisely one bounded context: ingestion, approval routing, currency normalization, analytics aggregation, budget forecasting, or report generation. Dependency bundles are minimized through tree-shaking build pipelines, maintaining cold-start package sizes below 8 MB per function to satisfy provider-imposed deployment artifact limits.

The NoSQL persistence layer utilizes a document-oriented store (compatible with MongoDB Atlas or AWS DynamoDB) with a composite partition key strategy: organization_id as the partition key and a timestamp-prefixed UUID as the sort key, enabling efficient range scans for monthly reporting without cross-partition scatter operations. The analytical aggregation records are maintained in a separate collection with pre-computed rollups at day, week, and month granularities to serve dashboard queries without scan-time aggregation overhead.

IV. EXPERIMENTAL SETUP

A. Tools and Technologies

The experimental implementation was deployed using the following technology stack. Backend runtime: Node.js 18 on AWS Lambda with 512 MB memory allocation per function. API management: AWS API Gateway (HTTP API mode) with JWT authorizer integration. Database: MongoDB Atlas M10 cluster (3-node replica set, ap-south-1 region). Frontend: React 18 with Recharts for analytical visualizations and Tailwind CSS for adaptive layout. CI/CD: GitHub Actions pipeline with automated unit testing (Jest), integration testing, and deployment to the target environment. Monitoring: AWS CloudWatch with custom metric emission from each function recording invocation count, duration, and error rate at one-minute granularity.

 This work is licensed under a Creative Commons Attribution 4.0 International License

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.2026.15662

B. Dataset Description

Experiments were conducted using a synthetic financial dataset generated to reflect realistic organizational expense patterns observed in mid-sized enterprise deployments. The dataset comprises 48,720 expense records spanning a 24-month simulated period across six functional departments: Engineering, Marketing, Sales, Operations, Human Resources, and Research & Development. Each record encodes: expense ID, submitter user ID, department, project code, expense category (14 categories including Travel, Accommodation, Software Licenses, Hardware, Catering, Training, Utilities, and others), amount in originating currency, receipt attachment reference, submission timestamp, approval chain status, and resolved base-currency amount.

Load testing employed the k6 open-source framework, generating synthetic API traffic at escalating concurrency levels: 10, 50, 100, 250, and 500 virtual users (VUs), each executing a standardized scenario comprising expense submission, approval action, and dashboard retrieval over a 5-minute sustained load window. Cold-start experiments isolated function initialization latency by invoking each function following a 30-minute idle period to guarantee complete container recycling.

C. Evaluation Metrics

System performance was assessed across four primary dimensions: (1) Latency: p50, p95, and p99 response time percentiles measured at the API gateway ingress point. (2) Throughput: Maximum sustained requests per second (RPS) achievable before the p95 latency exceeds a 500 ms service-level objective threshold. (3) Cost Efficiency: Estimated monthly invocation cost computed using provider pricing schedules at standardized workload volumes, normalized per 10,000 expense transactions. (4) Forecast Accuracy: Mean Absolute Percentage Error (MAPE) of the budget forecasting module computed on a held-out test set comprising the final 90 days of the synthetic dataset.

Additionally, cold-start frequency and duration were recorded per function type, and infrastructure provisioning overhead was compared against equivalent containerized (Docker on ECS Fargate) and virtual machine-based (EC2 t3.medium) deployments performing identical workloads.

V. RESULTS AND DISCUSSION

A. Latency Performance Analysis

Table II presents the measured API response latency statistics at each concurrency tier tested. The proposed serverless system maintained p50 latencies below 145 ms across all tested concurrency levels up to 250 VUs, rising to 198 ms at 500 VUs. The p95 latency remained below the 500 ms SLO threshold through 500 concurrent users, demonstrating robust performance under peak organizational load conditions.

TABLE II
API Response Latency (ms) at Varying Concurrency Levels

Concurrent Users	p50 Latency (ms)	p95 Latency (ms)	p99 Latency (ms)	Error Rate (%)
10	62	108	143	0.00
50	89	156	212	0.00
100	112	198	287	0.01
250	145	267	398	0.04
500	198	412	621	0.12

Cold-start latency was measured at an average of 812 ms for the initial invocation following a 30-minute idle period, with subsequent warm invocations averaging 94 ms. Provisioned concurrency pre-warming, applied to the Expense Ingestion and Dashboard Query functions during business hours (08:00–18:00 local time), eliminated cold-start occurrences for these high-frequency endpoints, effectively reducing observed cold-start events by 94% in production-representative workload simulations.

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.2026.15662

B. Throughput and Cost Efficiency

The system sustained a maximum throughput of 1,847 RPS before the p95 latency exceeded the 500 ms threshold, representing a 3.2× improvement over the equivalent containerized deployment (577 RPS) and a 6.8× improvement over the virtual machine-based baseline (272 RPS) when subjected to identical traffic profiles.

TABLE III
Comparative Infrastructure Cost Estimate per 10,000 Monthly Transactions

Deployment Model	Monthly Cost (USD)	Max Throughput (RPS)	Ops Overhead (hrs/mo)
VM-Based (EC2 t3.medium)	~$124.80	272	~18
Containerized (ECS Fargate)	~$68.40	577	~10
Proposed Serverless (FaaS)	**~$23.10**	**1,847**	**~3**

As illustrated in Table III, the proposed serverless framework achieved an estimated 81.5% cost reduction relative to the VM-based deployment and a 66.2% reduction relative to the containerized alternative at the reference workload volume. Operational overhead, measured as estimated engineer-hours required for routine infrastructure maintenance per month, decreased from approximately 18 hours (VM) and 10 hours (container) to approximately 3 hours under the serverless model, reflecting the elimination of OS patching, capacity planning, and scaling rule management responsibilities.

C. Budget Forecasting Accuracy

The exponential smoothing forecasting module was evaluated against the held-out 90-day test segment of the synthetic dataset. Table IV summarizes the forecast accuracy results by department, demonstrating that the model achieves an overall MAPE of 8.6%, corresponding to a forecast accuracy of 91.4%. Departments with consistent monthly expenditure patterns (Engineering, Operations) yield MAPE values as low as 5.2%, while departments with high expenditure variability (Marketing, Sales) exhibit higher error rates, motivating future investigation of seasonal decomposition methods for these categories.

TABLE IV
Budget Forecasting Accuracy by Department (MAPE, %)

Department	MAPE (%)	MAE (USD)	Forecast Horizon (days)
Engineering	5.2	214.30	30
Marketing	12.8	891.60	30
Sales	11.4	743.20	30
Operations	6.1	312.80	30
Human Resources	7.9	428.50	30
R&D	8.4	502.10	30
Overall	**8.6**	**515.40**	**30**

D. Anomaly Detection Performance

The Z-score anomaly detection module was validated against a curated set of 320 synthetically injected anomalous expense records representing inflated reimbursement claims, duplicate submissions, and out-of-policy category expenditures. The detection module achieved a precision of 87.2%, a recall of 84.5%, and an F1-score of 85.8%, demonstrating reliable identification of suspicious expense records while maintaining a manageable false-positive rate of 12.8% that does not materially burden the approval workflow.

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.2026.15662

E. Discussion of Findings

The experimental results collectively demonstrate that the proposed serverless expense management architecture delivers compelling advantages across all evaluated performance dimensions. The latency profile confirms that the event-driven, asynchronously decoupled design successfully insulates user-facing response times from back-end analytical processing workloads. The cost efficiency findings are particularly significant for organizations operating under constrained IT budgets: the pay-per-invocation model aligns infrastructure expenditure directly with usage volume, effectively eliminating the "waste floor" characteristic of always-provisioned deployments.

The forecasting accuracy results, while satisfactory for operational decision-support, reveal that high-variability departments would benefit from more sophisticated time-series models, such as ARIMA or Prophet, at the cost of increased function memory allocation and execution duration. This trade-off between forecasting fidelity and FaaS execution cost represents a domain-specific optimization opportunity warranting dedicated future investigation.

A notable limitation of the current implementation is its dependence on a single cloud provider's FaaS runtime, introducing potential vendor lock-in. The adoption of serverless framework abstraction layers (e.g., the Serverless Framework or AWS SAM) partially mitigates this concern by providing infrastructure-as-code portability, though runtime-specific optimizations would require re-tuning upon migration. Additionally, the current inter-function communication mechanism, relying on managed message queues, introduces a minimum analytical latency of approximately 800 ms from expense approval to dashboard reflection, which, while acceptable for financial reporting use cases, may necessitate architectural revision for near-real-time compliance monitoring scenarios.

VI. CONCLUSION AND FUTURE WORK

This paper has presented the design, implementation, and empirical evaluation of an intelligent serverless expense management system built upon Function-as-a-Service principles. The proposed architecture successfully addresses the limitations of conventional monolithic and containerized expense platforms by leveraging event-driven function composition, managed API gateway orchestration, and NoSQL persistence optimized for financial analytics workloads. Experimental results demonstrate that the system achieves sub-220 ms average API response latency, sustains throughput exceeding 1,800 RPS, reduces infrastructure costs by up to 81.5% relative to VM-based alternatives, and delivers budget forecasting with an overall MAPE of 8.6%.

The contributions of this research extend beyond the specific application domain: the architectural patterns, cold-start mitigation strategies, and RBAC-enabled multi-tenant data models presented herein are broadly applicable to any event-driven financial system requiring elastic scalability and predictable per-transaction cost profiles. The anomaly detection module, leveraging rolling Z-score thresholding within stateless FaaS functions, demonstrates that meaningful financial intelligence can be delivered without persistent server-side state.

Future research directions include: (1) Integration of transformer-based sequence models for improved expenditure forecasting on high-variance departmental budgets, deploying quantized model variants compatible with FaaS memory constraints. (2) Development of a multi-cloud portability layer enabling function deployment across AWS Lambda, Azure Functions, and Google Cloud Functions without provider-specific code modifications. (3) Extension of the approval workflow engine to support configurable multi-level authorization chains with delegated approval authorities and escalation timers. (4) Incorporation of optical character recognition (OCR) pipelines for automated receipt data extraction, reducing manual data entry overhead and improving expense record completeness. (5) Longitudinal study of real-world deployment cost trajectories across organizations of varying size to empirically validate the cost efficiency projections established in this controlled experimental setting.

REFERENCES

[1] D. Linthicum, "Cloud Computing and SOA Convergence in Your Enterprise," Addison-Wesley Professional, 2009.
[2] M. Roberts, "Serverless Architectures," Martin Fowler's Blog, Aug. 2018. [Online]. Available: https://martinfowler.com/articles/serverless.html
[3] E. Van Eyk, A. Iosup, S. Seif, and M. Thömmes, "The SPEC Cloud Group's Research Vision on FaaS and Serverless Architectures," in Proc. 2nd Int. Workshop on Serverless Computing (WoSC), 2017, pp. 1–4.
[4] A. Iosup et al., "A Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing," IEEE Trans. Parallel Distrib. Syst., vol. 22, no. 6, pp. 931–945, Jun. 2011.
[5] C. Richardson, Microservices Patterns: With Examples in Java. Manning Publications, 2018.
[6] I. Baldini et al., "Serverless Computing: Current Trends and Open Problems," in Research Advances in Cloud Computing, Springer, 2017, pp. 1–20.

 This work is licensed under a Creative Commons Attribution 4.0 International License

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.2026.15662

[7] H. Wang, X. Ma, and J. Liu, "A Microservices-Based Financial Transaction Management System," J. Cloud Comput., vol. 9, no. 1, pp. 1–15, 2020.
[8] S. Kumari and R. Pillai, "Automated Expense Management Using Rule-Based Classification in ERP Environments," Int. J. Enterprise Inform. Syst., vol. 16, no. 3, pp. 45–62, 2020.
[9] E. Jonas et al., "Cloud Programming Simplified: A Berkeley View on Serverless Computing," Tech. Rep. UCB/EECS-2019-3, Univ. of California, Berkeley, Feb. 2019.
[10] S. Eismann, J. Scheuner, E. Van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. Abad, and A. Iosup, "Serverless Applications: Why, When, and How?," IEEE Software, vol. 38, no. 1, pp. 32–39, Jan./Feb. 2021.
[11] J. Xu, Z. Chen, J. Tang, and S. Su, "T-Scheduler: For On-Line Dynamic VMScheduling in Large-Scale Heterogeneous Cloud Environment," in Proc. ACM SoCC, 2014, pp. 1–14.
[12] J. Nupponen and D. Taibi, "Serverless: What It Is, What to Do and What Not to Do," in Proc. IEEE Int. Conf. Software Architecture Companion (ICSA-C), 2020, pp. 49–52.
[13] S. Newman, Building Microservices: Designing Fine-Grained Systems, 2nd ed. O'Reilly Media, 2021.
[14] A. Fox et al., "Above the Clouds: A Berkeley View of Cloud Computing," Tech. Rep. UCB/EECS-2009-28, Univ. of California, Berkeley, 2009.
[15] R. Hyndman and G. Athanasopoulos, Forecasting: Principles and Practice, 3rd ed. OTexts, 2021. [Online]. Available: https://otexts.com/fpp3/
[16] M. Satyanarayanan, "The Emergence of Edge Computing," Computer, vol. 50, no. 1, pp. 30–39, Jan. 2017.
[17] CNCF, "CNCF Serverless Whitepaper v1.0," Cloud Native Computing Foundation, 2018. [Online]. Available: https://github.com/cncf/wg-serverless
[18] T. Lynn, P. Rosati, A. Lejeune, and V. Emeakaroha, "A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms," in Proc. IEEE CloudCom, 2017, pp. 162–169.
[19] Y. Gan et al., "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems," in Proc. ASPLOS, 2019, pp. 3–18.
[20] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The Rise of Serverless Computing," Commun. ACM, vol. 62, no. 12, pp. 44–54, Dec. 2019.

BIOGRAPHY

GUDELLI MOUNIKA received the B.Sc. degree from S.V.K.P. & Dr. K.S. Raju Arts and Science College, Penugonda, West Godavari, India, in 2022. 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, serverless architectures, cloud-native application development, financial technology systems, and software engineering. She is actively engaged in developing and studying modern cloud-based applications and distributed computing technologies.

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.

 This work is licensed under a Creative Commons Attribution 4.0 International License