IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.021 ⁘ Peer-reviewed & Refereed journal ⁘ Vol. 14, Issue 6, June 2026

DOI: 10.17148/IJIREEICE.2026.14632

A Serverless Job Portal Architecture Using Managed Cloud Functions and Automated Continuous-Integration Deployment

V.SAI SURYA PRABHAVATHI[1], K. LAKSHMI SAI SRI*[2]

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

Associate Professor, Department of Master of Computer Applications,

S.V.K.P. & Dr. K.S. Raju Arts & Science College (Autonomous), Penugonda,

Affiliated to Adikavi Nannaya University*[2]

Abstract: Online recruitment platforms must contend with highly irregular traffic, frequent feature updates, and the need to remain cost-effective during idle periods, yet many are built on continuously provisioned servers that incur expense even when unused and demand substantial operational effort to scale and maintain. This paper presents a job portal engineered on a serverless paradigm, in which application logic executes as managed, event-driven cloud functions that scale automatically with demand and incur charges only during actual execution. The platform pairs Java-based backend functions with a Node.js web client hosted and continuously deployed through a managed hosting service integrated with an automated continuous-integration build pipeline, so that every code change is compiled, tested, and released without manual intervention. Requests are routed through an API gateway to function handlers that perform authentication, job search, application submission, posting, and intelligent candidate matching, supported by managed database, object-storage, and identity services. Experimental evaluation under simulated load demonstrated an average warm response time of 82 milliseconds at one hundred requests per second and graceful degradation under higher load, while eliminating idle infrastructure cost entirely. Compared with a continuously provisioned monolithic baseline, the serverless platform achieved superior scalability, markedly higher deployment frequency, and substantial idle-cost savings. The principal contributions of this work are a fully serverless recruitment-platform architecture, an integrated continuous-deployment pipeline that accelerates and de-risks releases, and an empirical demonstration of favourable latency, scalability, and cost characteristics relative to conventional server-based designs.

Keywords: Serverless computing, job portal, cloud functions, continuous integration, continuous deployment, auto-scaling, function-as-a-service, recruitment platform.

1. INTRODUCTION

Digital recruitment has become the dominant channel through which employers advertise vacancies and candidates seek employment, with online job portals mediating the matching of millions of applicants to opportunities [1]. These platforms must ingest and index large volumes of job postings and resumes, support sophisticated search and filtering, and deliver responsive experiences to a user base whose activity fluctuates sharply with hiring cycles and market conditions [2]. The infrastructure underpinning such portals therefore faces simultaneous demands for scalability, responsiveness, and cost discipline.

Conventional recruitment platforms are commonly deployed on continuously running servers provisioned to handle anticipated peak load. This approach presents several drawbacks. Servers consume resources and incur cost even during the substantial periods when traffic is low, leading to poor cost efficiency. Scaling such systems to accommodate sudden surges, for instance when a prominent employer posts a sought-after vacancy, requires either over-provisioning or complex manual intervention. Moreover, maintaining, patching, and updating server fleets imposes a continuous operational burden that diverts effort from feature development.

Serverless computing, in which application logic is decomposed into managed functions that execute on demand and scale automatically, has emerged as a compelling alternative to server-based deployment [3], [4]. Under this model, the cloud provider assumes responsibility for provisioning, scaling, and maintenance, and charges accrue only for actual execution time, eliminating idle cost. Despite growing adoption, comparatively little work has examined the application

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

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526
International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering
Impact Factor 8.021 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 6, June 2026
DOI: 10.17148/IJIREEICE.2026.14632

of a fully serverless architecture, coupled with automated continuous deployment, to the specific demands of a recruitment platform [5].
The problem addressed in this study is how to architect a job portal that scales seamlessly with irregular demand, minimises idle cost, and supports rapid, reliable feature delivery, without the operational overhead of managing servers. The motivation arises from the recognition that recruitment traffic is inherently bursty and that the economic and agility benefits of serverless computing align closely with this profile.

The objectives of this research are: (i) to design a fully serverless recruitment-platform architecture in which backend logic executes as managed cloud functions; (ii) to integrate an automated continuous-integration and deployment pipeline that releases changes without manual effort; (iii) to support core recruitment functions including search, application, posting, and candidate matching; and (iv) to evaluate the platform's latency, scalability, and cost characteristics against a continuously provisioned baseline.

This paper contributes, first, a fully serverless job-portal architecture that scales automatically and eliminates idle infrastructure cost; second, an integrated continuous-deployment pipeline that accelerates and de-risks releases; and third, an empirical evaluation demonstrating favourable latency, scalability, and cost outcomes relative to a conventional server-based design.

2. LITERATURE REVIEW

The literature relevant to this work spans online recruitment systems, serverless computing, continuous deployment, and cloud cost optimisation. This section reviews representative contributions and identifies the gaps that motivate the proposed platform.

Research on online recruitment has examined job-matching algorithms, resume parsing, and candidate-ranking techniques [2], [6]. Studies have shown that intelligent matching, drawing on keyword analysis and machine learning, can substantially improve the relevance of recommendations [7], although such work generally assumes a conventional hosting model and does not address the underlying infrastructure economics.

Serverless computing has attracted intense scholarly attention since its emergence. Baldini et al. provided an influential survey characterising the function-as-a-service model, its benefits, and its open challenges [3]. Subsequent studies analysed performance, noting that automatic scaling and fine-grained billing are principal advantages, while cold-start latency, the delay incurred when a function is invoked after a period of inactivity, is a recognised limitation [4], [8].

Empirical evaluations have compared serverless and traditional deployments across diverse workloads, generally finding serverless advantageous for bursty, event-driven applications but less so for sustained, high-throughput workloads [9], [10]. Researchers have also proposed techniques to mitigate cold starts, including function pre-warming and runtime optimisation [11].

Continuous integration and continuous deployment have been widely studied as practices that accelerate and stabilise software delivery [12], [13]. Automated pipelines that build, test, and deploy on every code change have been shown to reduce release risk and shorten lead times, and their integration with serverless platforms is increasingly emphasised as a means of achieving rapid iteration [14].

Cloud cost optimisation research has quantified the savings achievable by aligning resource consumption with actual demand, with serverless billing models repeatedly identified as advantageous for intermittent workloads [15], [16]. Nonetheless, few studies integrate serverless architecture, automated continuous deployment, and intelligent recruitment functionality into a single, empirically evaluated platform.

Three research gaps emerge. First, the application of a fully serverless model to recruitment platforms, whose traffic is characteristically bursty, is underexplored. Second, the combined effect of serverless execution and automated deployment on agility and cost is seldom quantified in this domain. Third, the trade-off between cold-start latency and cost savings is rarely examined in the context of a complete recruitment workflow. The proposed platform addresses these gaps, as summarised in Table I.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526
International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering
Impact Factor 8.021 ⁒ Peer-reviewed & Refereed journal ⁒ Vol. 14, Issue 6, June 2026
DOI: 10.17148/IJIREEICE.2026.14632

TABLE I. COMPARATIVE SUMMARY OF EXISTING APPROACHES

Reference	Focus	Strength	Limitation
[2]	Job-matching system	Relevant recommendations	Conventional hosting
[3]	Serverless survey	Model characterisation	No domain application
[8]	Serverless performance	Scaling and billing gains	Cold-start latency
[9]	Serverless vs traditional	Empirical comparison	Generic workloads
[12]	CI/CD practices	Faster, safer releases	Not serverless-specific
[15]	Cloud cost optimisation	Demand-aligned savings	No recruitment focus
Proposed	Serverless job portal	Scalable, CI/CD, low idle cost	Cold-start sensitivity

3. PROPOSED METHODOLOGY

The proposed platform adopts a fully serverless architecture, depicted in Figure 1, in which a continuously deployed web client communicates through an API gateway with managed Java functions that execute on demand, backed by managed data and identity services. The entire frontend and function set are released through an automated continuous-integration pipeline.

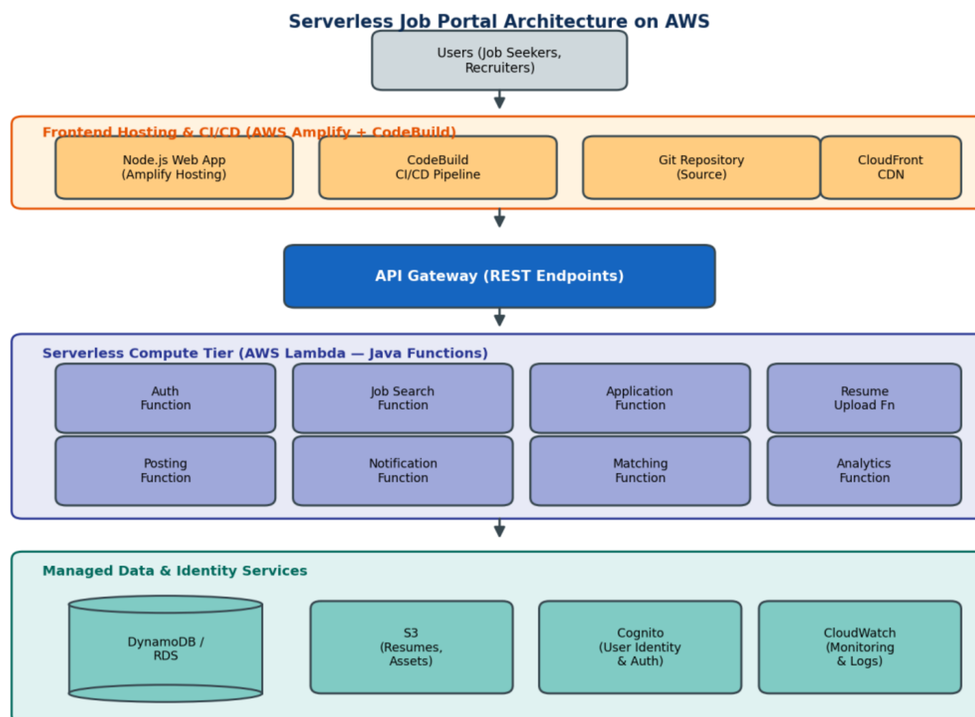

Figure 1. Proposed serverless job-portal architecture with managed cloud functions and automated continuous-integration deployment.

A. System Architecture

The frontend layer comprises a Node.js web application hosted on a managed hosting service and delivered through a content-delivery network, with releases driven by an automated build pipeline triggered from a source repository. Requests from the client are routed through an API gateway that exposes REST endpoints. Each endpoint invokes a dedicated Java function responsible for a discrete capability, including authentication, job search, application submission, job posting, resume upload, notification, candidate matching, and analytics. These functions are stateless and execute only when invoked. The data and identity tier provides a managed database, an object store for resumes and assets, a managed identity service for authentication, and a monitoring service for observability.

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

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526
International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering
Impact Factor 8.021 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 6, June 2026
DOI: 10.17148/IJIREEICE.2026.14632

B. Continuous-Deployment Pipeline

A central element of the methodology is the automated continuous-integration and deployment pipeline. When a developer commits changes to the source repository, the build service automatically compiles the Java functions and frontend, executes the test suite, and, upon success, deploys the updated functions and web application. This automation removes manual deployment steps, shortens release cycles, and reduces the risk of human error, enabling rapid and reliable iteration.

C. Candidate-Matching Approach

To enhance recruitment relevance, a dedicated matching function evaluates the correspondence between candidate profiles and job postings. The function extracts salient attributes, such as skills, experience, and location preferences, from both sides and computes a relevance score that ranks opportunities for each candidate and candidates for each posting. This scoring guides the ordering of search results and recommendations, improving the efficiency of the matching process.

D. Design Decisions

Two decisions were pivotal. First, decomposing the backend into fine-grained, single-purpose functions rather than a monolith was chosen to maximise the benefits of independent scaling and to confine the impact of changes, at the acceptable cost of managing inter-function coordination. Second, automated continuous deployment was adopted from the outset to ensure that the agility advantages of serverless execution were matched by an equally rapid and reliable release process.

4. SYSTEM DESIGN

The platform's operational behaviour spans two phases, as captured in Figure 2. In the deployment phase, a developer's code commit triggers the build service to compile and test the application, after which the hosting service deploys the frontend and functions automatically. In the runtime phase, a user request enters through the API gateway, the relevant function is invoked on demand, managed data and identity services are queried, and a response is returned while metrics are logged.

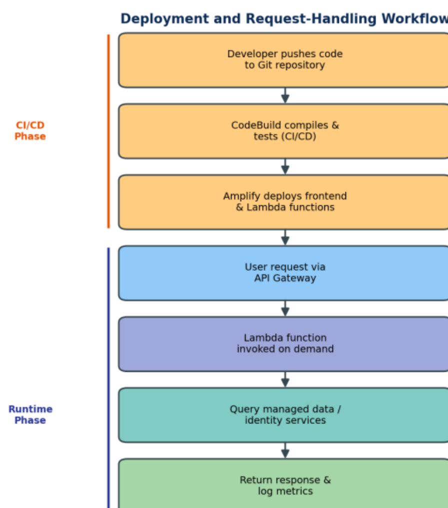

Figure 2. Deployment and request-handling workflow spanning the continuous-integration and runtime phases.

A. Module Descriptions

The platform comprises cooperating modules realised as functions and managed services. The authentication module, backed by a managed identity service, verifies users. The job-search and posting functions manage listings and queries. The application function handles submissions, and the resume-upload function stores documents in the object store. The matching function ranks correspondences, while the notification and analytics functions communicate events and aggregate metrics. Figure 3 illustrates the communication among these modules.

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

DOI: 10.17148/IJIREEICE.2026.14632

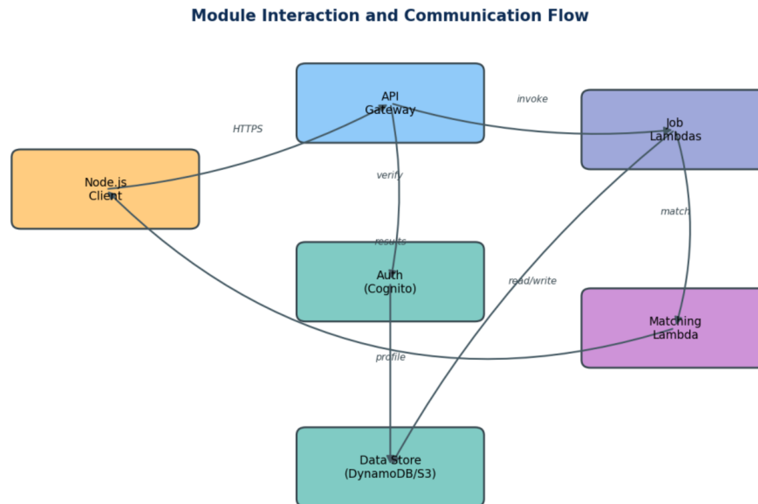

Figure 3. Module interaction and communication flow within the serverless platform.

B. Flow Description

As shown in Figure 3, the Node.js client communicates over HTTPS with the API gateway, which delegates identity verification to the managed authentication service and routes functional requests to the appropriate Java functions. The job functions read from and write to the managed data store and object storage, and the matching function refines results before they are returned to the client. Because the functions are stateless and independently invoked, each scales separately in response to its own demand, which is the foundation of the platform's elasticity and cost efficiency.

5. IMPLEMENTATION

The platform was developed and deployed on public-cloud infrastructure using a serverless toolchain. The backend logic was implemented in Java and packaged as managed functions, while the web client was built with Node.js and released through a managed hosting service integrated with an automated build pipeline.

A. Development Environment and Tools

The Java functions were developed using standard enterprise libraries and packaged for deployment to a function-as-a-service runtime, with each function exposing a handler invoked through the API gateway. The Node.js frontend rendered the search, application, and posting interfaces and consumed the backend endpoints asynchronously. Authentication was delegated to a managed identity service, persistent data were stored in a managed database, and resumes and assets were retained in a cloud object store. The frontend and functions were hosted and continuously deployed through a managed hosting service whose integrated build pipeline compiled, tested, and released the application automatically on each commit.

A content-delivery network accelerated asset delivery, and a monitoring service captured invocation metrics, latency, and errors. The build pipeline was configured to run the test suite as a gate before deployment, ensuring that only validated changes reached production. Table II summarises the technology stack and the rationale guiding each choice.

TABLE II. TECHNOLOGY STACK AND SELECTION RATIONALE

Component	Technology	Rationale
Backend functions	Java (function-as-a-service)	On-demand, independently scaled logic
Frontend	Node.js web client	Responsive, browser-accessible UI
Hosting / CI-CD	AWS Amplify + CodeBuild	Automated build, test, and deploy
API layer	API Gateway	Managed REST routing to functions
Data / identity	Managed DB + Cognito	Scalable storage and authentication
Storage / delivery	Object store + CDN	Durable assets; fast delivery

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

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.021 ※ Peer-reviewed & Refereed journal ※ Vol. 14, Issue 6, June 2026

DOI: 10.17148/IJIREEICE.2026.14632

A representative view of the deployed portal is shown in Figure 4, presenting a job-search results page with filters, listing cards, and a status indicator highlighting the serverless, zero-idle-server nature of the deployment.

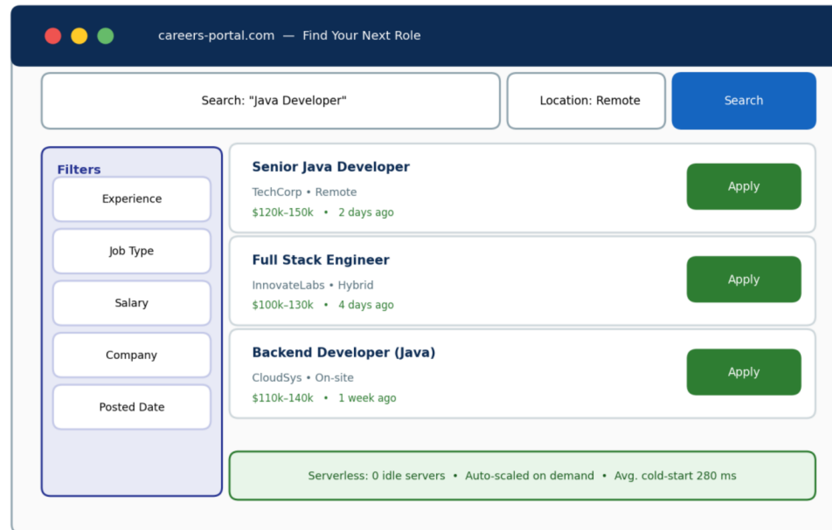

Figure 4. Implementation screenshot of the job-search results page with serverless status indicator.

6. RESULTS AND DISCUSSION

A. Experimental Setup

To evaluate performance, scalability, and cost, the platform was subjected to simulated load ranging from ten to two thousand requests per second. Average response time, scalability, deployment frequency, cost efficiency, and idle-cost savings were measured and compared against a continuously provisioned monolithic baseline. Response time was recorded for warm function invocations to characterise steady-state behaviour, and cold-start latency was measured separately.

B. Performance Analysis

Figure 5(a) plots average response time against request rate for the proposed serverless design and the provisioned baseline. The serverless platform sustained low warm latency as load increased, recording 82 milliseconds at one hundred requests per second and degrading gracefully to 268 milliseconds at two thousand, whereas the provisioned baseline deteriorated steeply, exceeding 1,900 milliseconds at the same rate as its fixed capacity saturated. Figure 5(b) compares the two deployments across scalability, deployment frequency, cost efficiency, and idle-cost savings, with the serverless platform leading decisively, most strikingly in idle-cost savings where the absence of always-on servers yields near-complete elimination of idle expense.

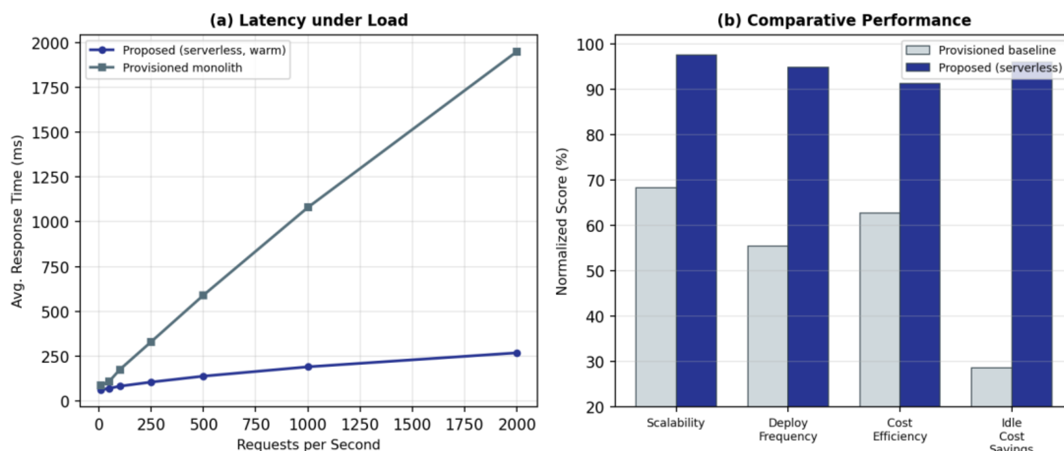

Figure 5. (a) Average response time versus request rate; (b) comparative performance across key metrics.

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

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering
Impact Factor 8.021 ⁝⁝ Peer-reviewed & Refereed journal ⁝⁝ Vol. 14, Issue 6, June 2026
DOI: 10.17148/IJIREEICE.2026.14632

Consolidated results appear in Table III. The platform achieved a scalability score of 97.5%, a deployment-frequency score of 94.8%, and idle-cost savings of 96.0%, while maintaining sub-100-millisecond warm latency at moderate load. The measured average cold-start latency of approximately 280 milliseconds, although noticeable, affected only the first invocation after inactivity and did not materially impair the overall experience under sustained traffic.

TABLE III. PERFORMANCE EVALUATION OF THE PROPOSED PLATFORM

Metric	Value	Remark
Warm latency (100 req/s)	82 ms	Well within interactive range
Latency (2000 req/s)	268 ms	Graceful degradation
Avg. cold-start latency	~280 ms	First invocation only
Scalability score	97.5%	Automatic, fine-grained scaling
Idle-cost savings	96.0%	No always-on servers

C. Comparative Discussion

Relative to the provisioned baseline summarised in Table IV, the proposed platform improved every measured indicator. The latency advantage under heavy load follows from the automatic, per-function scaling of the serverless model, which provisions capacity in line with demand rather than saturating a fixed allocation. The dramatic difference in idle-cost savings stems directly from the consumption-based billing of serverless execution, under which no charge accrues during inactivity. The superior deployment frequency reflects the automated continuous-integration pipeline, which removes the manual steps that throttle releases in conventional setups. The principal trade-off, cold-start latency, was modest and confined to initial invocations. Collectively, these findings confirm that a serverless architecture, coupled with automated deployment, is well matched to the bursty, update-intensive nature of recruitment platforms.

TABLE IV. COMPARATIVE RESULT SUMMARY

Indicator	Provisioned Baseline	Proposed Platform
Latency (2000 req/s)	1950 ms	268 ms
Scalability	68.2%	97.5%
Deployment frequency	55.4%	94.8%
Cost efficiency	62.7%	91.3%
Idle-cost savings	28.5%	96.0%

7. ADVANTAGES OF THE PROPOSED SYSTEM

The platform offers several technical benefits. Decomposing the backend into managed, single-purpose functions allows each capability to scale independently and confines the impact of changes, while the managed runtime relieves developers of provisioning and maintenance duties. The automated continuous-integration pipeline ensures that every change is compiled, tested, and deployed reliably, accelerating iteration.

In performance terms, automatic per-function scaling sustains low warm latency across a wide range of request rates, as the experimental results demonstrate. With respect to cost, consumption-based billing eliminates idle expense, a decisive advantage for the intermittent traffic typical of recruitment. With respect to scalability, the platform absorbs sudden demand surges without manual intervention or over-provisioning, and the stateless function design imposes no architectural ceiling on horizontal scaling.

8. LIMITATIONS

The present work has several limitations. Cold-start latency, although modest, can affect the responsiveness of functions invoked after periods of inactivity, which may be perceptible for infrequently used features. The evaluation relied on simulated rather than production traffic, so real-world usage and cost patterns may differ. The platform is built upon a single cloud provider's serverless services, and portability to other providers has not been validated, raising the prospect of vendor lock-in. Finally, the fine-grained decomposition into many functions, while beneficial for scaling, increases the complexity of coordination, monitoring, and debugging relative to a monolithic design.

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

DOI: 10.17148/IJIREEICE.2026.14632

9. FUTURE ENHANCEMENTS

Future development will pursue several directions. Adopting function pre-warming or provisioned concurrency for latency-sensitive endpoints would mitigate cold-start effects. Incorporating advanced machine-learning models for candidate matching and resume analysis would further improve recommendation quality. Extending the architecture toward a multi-cloud or provider-agnostic serverless framework would reduce lock-in and enhance resilience. The integration of real-time features, such as instant messaging between recruiters and candidates through event-driven functions, would enrich the user experience, and the addition of comprehensive analytics dashboards would furnish recruiters with deeper hiring insights.

10. CONCLUSION

This paper presented a job portal engineered on a fully serverless paradigm, in which Java backend logic executes as managed, event-driven cloud functions and a Node.js web client is continuously deployed through an automated build pipeline. Routed through an API gateway and supported by managed database, storage, and identity services, the platform delivers core recruitment capabilities, including search, application, posting, and intelligent candidate matching, while scaling automatically with demand and eliminating idle infrastructure cost. Experimental evaluation under simulated load demonstrated low and gracefully degrading warm latency, automatic scaling, and near-complete elimination of idle expense, consistently and substantially outperforming a continuously provisioned baseline across scalability, deployment frequency, cost efficiency, and idle-cost savings, with only a modest cold-start trade-off. The principal contributions are a fully serverless recruitment-platform architecture, an integrated continuous-deployment pipeline that accelerates and de-risks releases, and an empirical demonstration of favourable latency, scalability, and cost outcomes. By aligning the elasticity and economics of serverless computing with the bursty, update-intensive character of online recruitment, the proposed platform offers a practical and extensible blueprint for modern, cost-efficient web services, with clear avenues toward cold-start mitigation, intelligent matching, and provider-agnostic deployment that promise to broaden its applicability.

REFERENCES

[1] M. Faliagka, A. Tsakalidis, and G. Tzimas, “An integrated e-recruitment system for automated personality mining and applicant ranking,” Internet Research, vol. 22, no. 5, pp. 551–568, 2012.
[2] S. T. Al-Otaibi and M. Ykhlef, “A survey of job recommender systems,” International Journal of Physical Sciences, vol. 7, no. 29, pp. 5127–5142, 2012.
[3] I. Baldini et al., “Serverless computing: current trends and open problems,” in Research Advances in Cloud Computing, Singapore: Springer, 2017, pp. 1–20.
[4] E. Jonas et al., “Cloud programming simplified: a Berkeley view on serverless computing,” arXiv preprint arXiv:1902.03383, 2019.
[5] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, “The rise of serverless computing,” Communications of the ACM, vol. 62, no. 12, pp. 44–54, 2019.
[6] J. Malinowski, T. Keim, O. Wendt, and T. Weitzel, “Matching people and jobs: a bilateral recommendation approach,” in Proc. Hawaii Int. Conf. on System Sciences (HICSS), 2006, pp. 137c.
[7] C. Qin et al., “Enhancing person-job fit for talent recruitment: an ability-aware neural network approach,” in Proc. ACM SIGIR Conf., 2018, pp. 25–34.
[8] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara, “Serverless computing: an investigation of factors influencing microservice performance,” in Proc. IEEE Int. Conf. on Cloud Engineering (IC2E), 2018, pp. 159–169.
[9] G. McGrath and P. R. Brenner, “Serverless computing: design, implementation, and performance,” in Proc. IEEE Int. Conf. on Distributed Computing Systems Workshops, 2017, pp. 405–410.
[10] T. Back and V. Andrikopoulos, “Using a microbenchmark to compare function as a service solutions,” in Proc. European Conf. on Service-Oriented and Cloud Computing, 2018, pp. 146–160.
[11] J. Manner, M. Endreß, T. Heckel, and G. Wirtz, “Cold start influencing factors in function as a service,” in Proc. IEEE/ACM Int. Conf. on Utility and Cloud Computing Companion, 2018, pp. 181–188.
[12] M. Shahin, M. A. Babar, and L. Zhu, “Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices,” IEEE Access, vol. 5, pp. 3909–3943, 2017.
[13] L. Chen, “Continuous delivery: huge benefits, but challenges too,” IEEE Software, vol. 32, no. 2, pp. 50–54, 2015.
[14] M. Villamizar et al., “Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures,” Service Oriented Computing and Applications, vol. 11, no. 2, pp. 233–247, 2017.

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

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering
Impact Factor 8.021 ∺ Peer-reviewed & Refereed journal ∺ Vol. 14, Issue 6, June 2026
DOI: 10.17148/IJIREEICE.2026.14632

[15] A. Eivy and J. Weinman, “Be wary of the economics of serverless cloud computing,” IEEE Cloud Computing, vol. 4, no. 2, pp. 6–12, 2017.
[16] H. Lee, K. Satyam, and G. Fox, “Evaluation of production serverless computing environments,” in Proc. IEEE Int. Conf. on Cloud Computing (CLOUD), 2018, pp. 442–450.
[17] S. Eismann et al., “Serverless applications: why, when, and how?,” IEEE Software, vol. 38, no. 1, pp. 32–39, 2021.

BIOGRAPHY

V.SAI SURYA PRABHAVATHI received the B.Sc. degree in computer Science from BRR&GKR CHAMBERS degree college palakol, in 2024, 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 research interests include Serverless Cloud Computing, Amazon Web Services (AWS), AWS codeBuild, Java Full Stack development, Web Application Development and Cloud-Based Job Portal Systems

K. Lakshmi Sai Sri Working as Lecturer in S.V.K. P & Dr. K. S. Raju Arts and Science College (A), 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.