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.2026.14635

A Cloud-Native Hospital Appointment Scheduling and Electronic Health Record Management System Leveraging AWS Services and DevOps Automation

KADALI VASANTH[1], Smt A.N. RAMA MANI*[2]

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

Associate Professor, Department of Master of Computer Applications, S.V.K.P & Dr. K.S. Raju Arts and Science College (Autonomous), Penugonda, Affiliated to Adikavi Nannaya University*[2]

*Corresponding Author

Abstract: The growing demand for accessible and reliable digital healthcare has exposed the limitations of conventional, on-premise hospital management software, which frequently suffers from poor scalability, fragmented patient data, and unreliable service availability during peak load. This study presents the design and evaluation of a cloud-native platform that unifies outpatient appointment scheduling with electronic health record (EHR) management while embedding continuous integration and continuous delivery (CI/CD) practices throughout its lifecycle. The proposed system adopts a containerized microservice architecture deployed on Amazon Web Services (AWS), in which application logic is implemented in Python and the presentation layer is built using Node.js. A priority-aware scheduling routine allocates consultation slots, while patient records are persisted across managed relational and object storage services to balance consistency and elasticity. Infrastructure provisioning, automated testing, and deployment are orchestrated through an Infrastructure-as-Code and pipeline-driven DevOps workflow. Experimental evaluation under synthetic concurrent load demonstrates that the platform sustains an average response latency of roughly 312 ms at 1000 simultaneous users—markedly lower than a monolithic baseline—while horizontal auto-scaling preserves a measured service availability of 99.8%. Adoption of automated pipelines reduced deployment lead time from approximately 95 minutes to under 10 minutes and shortened mean recovery time after failure. The principal contributions are an integrated appointment-plus-EHR reference architecture, a reproducible DevOps automation strategy for healthcare workloads, and an empirical performance characterization that quantifies the benefits of cloud elasticity for clinical service delivery.

Keywords: Cloud computing; Electronic Health Records; Appointment scheduling; Amazon Web Services; DevOps; Microservices; Continuous integration; Healthcare informatics

1. INTRODUCTION

The digitization of healthcare delivery has accelerated over the past decade, driven by rising patient expectations, regulatory mandates for electronic recordkeeping, and the proven operational gains of computerized clinical workflows. Hospitals and outpatient clinics increasingly rely on software platforms to coordinate consultations, maintain longitudinal patient histories, and exchange diagnostic information among care teams. Despite this progress, a substantial proportion of healthcare institutions, particularly in resource-constrained settings, continue to operate legacy systems that were never designed for the scale, availability, and security demands of contemporary digital health.

Traditional hospital information systems are commonly deployed on fixed, on-premise infrastructure. Such deployments couple the application tightly to a finite pool of servers, which leads to degraded performance when appointment traffic spikes—for example, during seasonal illness surges or public health emergencies. Equally problematic is the fragmentation of patient data across disconnected modules, which obstructs the formation of a coherent longitudinal record and complicates clinical decision-making. Manual release and maintenance procedures further increase the risk of prolonged downtime, configuration drift, and human error.

Problem statement. The central problem addressed in this work is the absence of an integrated, elastically scalable, and operationally automated platform that simultaneously handles outpatient appointment scheduling and electronic health

DOI: 10.17148/IJIREEICE.2026.14635

record management while guaranteeing high availability and rapid, low-risk software delivery. Existing solutions tend to optimize one dimension—either scheduling convenience or record management—without treating reliability and deployment automation as first-class architectural concerns.

Motivation. Cloud computing offers on-demand elasticity, managed data services, and pay-as-you-go economics that are well matched to the bursty and unpredictable nature of healthcare demand. When combined with DevOps practices, where infrastructure is codified and releases are automated, these capabilities can dramatically reduce operational toil and improve resilience. This convergence motivates a unified design that treats scalability, data integrity, and continuous delivery as inseparable goals rather than afterthoughts.

Research objectives. The objectives of this study are: (i) to design a modular, cloud-native architecture that integrates appointment scheduling and EHR management; (ii) to implement the platform using Python services and a Node.js front end deployed on AWS; (iii) to embed an automated CI/CD and Infrastructure-as-Code workflow; and (iv) to empirically evaluate performance, scalability, and deployment efficiency against a conventional baseline.

Contributions. This paper makes the following contributions. First, it proposes a reference architecture that cohesively couples scheduling and clinical records over managed AWS services. Second, it details a reproducible DevOps automation pipeline tailored to a regulated healthcare workload. Third, it provides a quantitative evaluation demonstrating substantial improvements in latency, availability, and release velocity. The remainder of the paper is organized as follows. Section 2 reviews related work; Section 3 presents the proposed methodology; Section 4 describes the system design; Section 5 details the implementation; Section 6 reports results and discussion; Sections 7 through 9 analyze advantages, limitations, and future work; and Section 10 concludes.

2. LITERATURE REVIEW

Research on digital healthcare platforms spans appointment management, electronic health records, cloud migration, and the operationalization of software delivery. This section surveys representative contributions and identifies the gaps that the present work seeks to close.

Early efforts to computerize appointment booking focused on reducing patient wait times and no-show rates through web and mobile interfaces. Several studies demonstrated that online self-scheduling improves utilization of clinical resources, yet many of these systems were monolithic and lacked mechanisms to absorb sudden demand. Subsequent work on electronic health records emphasized interoperability and standardized data exchange, with health information exchange frameworks promoting structured formats to enable cross-provider continuity of care [1], [2]. However, these record-centric systems often treated scheduling as an external concern.

The migration of clinical systems to the cloud has attracted sustained attention. Investigations into cloud-based health information systems report gains in availability and cost efficiency, while cautioning that data privacy and regulatory compliance remain decisive constraints [3], [4]. Studies of managed database and storage services have shown that separating structured records from large binary artifacts, such as medical imaging, improves both performance and cost predictability [5]. Complementary research on access control and encryption underscores the importance of identity management and key governance in protecting sensitive health data [6].

Architecturally, the shift from monolithic applications toward microservices has been widely studied as a means of improving maintainability and independent scalability. Comparative analyses indicate that microservice decomposition, when paired with containerization, enables fine-grained scaling and fault isolation, although it introduces operational complexity in service coordination and observability [7], [8]. Container orchestration platforms have been evaluated as effective substrates for elastic healthcare services, provided that monitoring and automated recovery are in place [9].

A parallel body of work addresses DevOps and continuous delivery. Empirical research links automated pipelines and Infrastructure-as-Code to higher deployment frequency, shorter lead times, and faster recovery, as captured by widely cited delivery-performance metrics [10], [11]. Within healthcare specifically, recent studies argue that automated, auditable deployment processes can support compliance by making changes traceable and reproducible [12]. Scheduling optimization has likewise been explored through heuristic and priority-based algorithms that balance clinician availability against patient urgency [13]. More recent contributions examine serverless and event-driven designs for notification and asynchronous workflows in clinical settings [14], [15].

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.2026.14635

Research gaps. Across this literature, three gaps recur. First, scheduling and I functions are seldom co-designed within a single elastic architecture. Second, many cloud healthcare proposals describe deployment topologies but omit a concrete, reproducible automation pipeline. Third, quantitative, head-to-head evaluations of latency, availability, and delivery metrics against a conventional baseline are comparatively rare. Table I summarizes representative works against these dimensions and positions the present study.

3. PROPOSED METHODOLOGY

3.1 System Architecture Overview

The proposed platform follows a layered, cloud-native architecture composed of a client tier, an edge and API-gateway tier, a containerized application tier, and a managed data tier, all underpinned by a cross-cutting DevOps automation layer. Figure 1 depicts the overall organization. Requests originate from patient, clinician, and administrative clients and traverse a content delivery network and an authenticated API gateway before reaching the appropriate microservice. This separation of concerns allows each tier to scale and evolve independently.

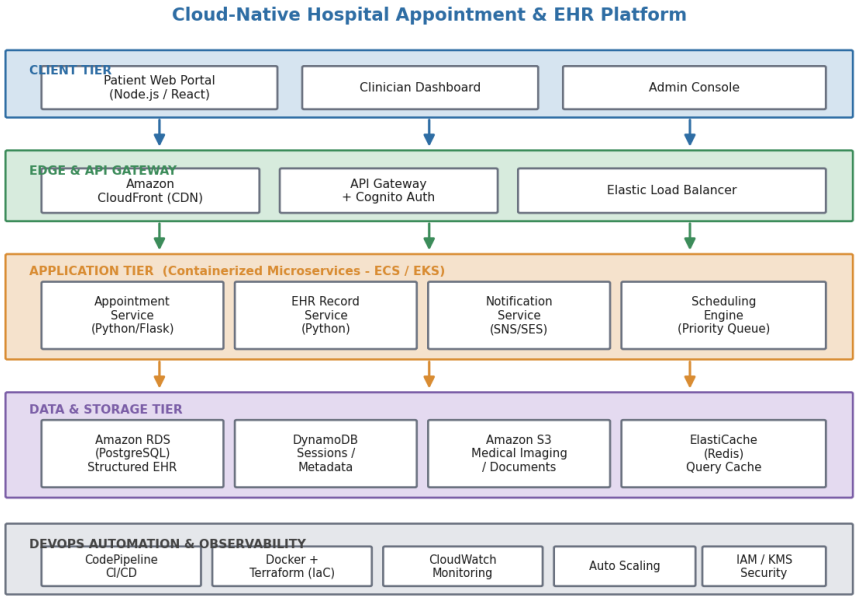

Fig. 1. Proposed cloud-native system architecture showing the client, edge, application, and data tiers with the cross-cutting DevOps automation layer. (Placement: top of this section.)

3.2 Technologies Used

Application services are implemented in Python, chosen for its mature ecosystem of web frameworks and data-handling libraries and its suitability for rapid, maintainable backend development. The presentation layer is built with Node.js, enabling a responsive, event-driven interface for booking and record access. AWS provides the deployment substrate: compute is delivered through a container service, persistent structured data resides in a managed relational database, unstructured artifacts are stored in object storage, and an in-memory cache accelerates frequent read operations. Identity, encryption, monitoring, and message delivery are handled by their respective managed services.

3.3 Scheduling Algorithm

Appointment allocation is governed by a priority-aware scheduling routine. Each booking request is characterized by the requested specialty, patient-supplied urgency, and the availability calendar of eligible clinicians. Candidate slots are inserted into a priority queue ordered by a composite key combining clinical urgency and temporal proximity. The routine selects the highest-priority feasible slot, atomically reserves it within a transactional boundary to prevent double-booking, and, when no slot is feasible, places the request on a waitlist queue that is re-evaluated as cancellations occur. This design keeps allocation deterministic and auditable while remaining responsive under contention.

3.4 Workflow and Design Decisions

Figure 2 illustrates the end-to-end booking workflow. After authentication, a patient searches by specialty, the scheduling engine proposes an optimal slot, and a confirmed booking triggers a durable write to the relational store together with an

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.2026.14635

asynchronous notification. Several deliberate design decisions shape the system: managed services are favored over self-hosted components to minimize operational burden; stateless application containers enable horizontal scaling; and the strict separation of structured records from large binary objects optimizes both query performance and storage cost. Crucially, every infrastructure component is expressed as code so that environments are reproducible and auditable.

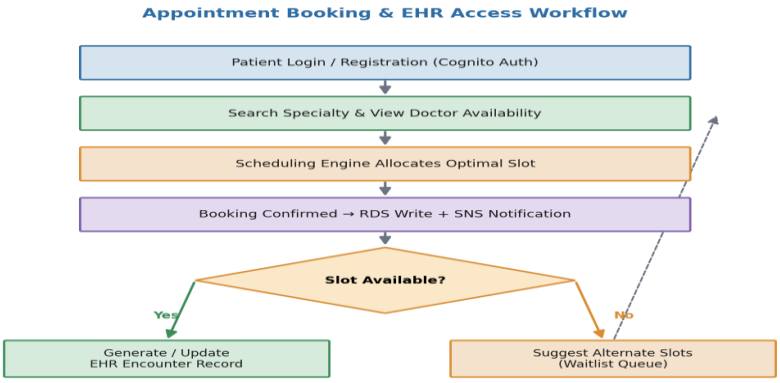

Fig. 2. Workflow diagram of the appointment booking and EHR access process, including slot-availability decision logic and waitlisting. (Placement: end of Section 3.)

4. SYSTEM DESIGN

4.1 Architectural Decomposition

The application tier is decomposed into cohesive microservices: an appointment service that manages booking lifecycles, an EHR service that governs clinical records, a notification service for asynchronous messaging, and a scheduling engine that encapsulates the allocation logic described earlier. Each service exposes a well-defined interface through the API gateway and communicates with the data tier via a dedicated data-access layer. This decomposition isolates failures, permits independent deployment, and allows the most heavily used services to scale without over-provisioning the entire system.

4.2 Module Descriptions

The authentication module integrates with a managed identity provider to enforce role-based access for patients, clinicians, and administrators. The appointment module coordinates with the scheduling engine and persists confirmed bookings. The EHR module manages structured encounter records and links to imaging artifacts in object storage. The notification module dispatches confirmations and reminders through managed messaging channels. A shared data-access layer abstracts persistence concerns and enforces transactional integrity across stores.

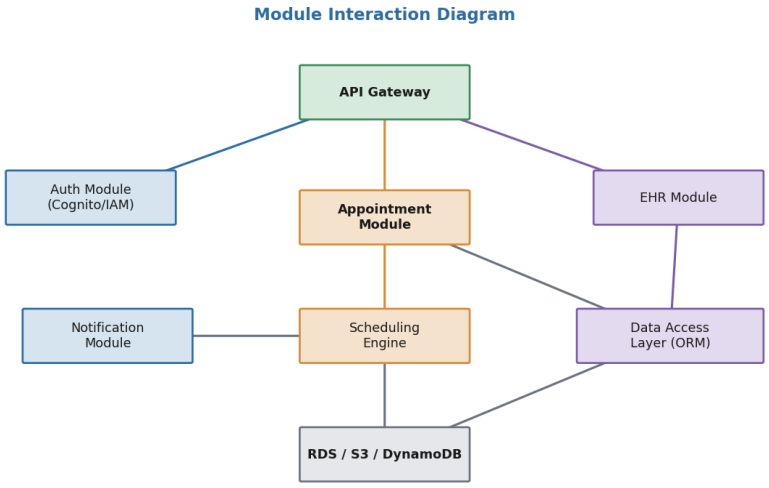

Fig. 3. Module interaction diagram depicting communication paths among the gateway, authentication, appointment, EHR, scheduling, notification, and data-access modules. (Placement: within Section 4.2.)

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.2026.14635

4.3 Data Flow

As shown in Figure 3, inbound requests are authenticated at the gateway and routed to the relevant module. The appointment and EHR modules delegate persistence to the data-access layer, which transparently directs structured writes to the relational database, session and metadata operations to the key-value store, and large objects to object storage. Read-heavy paths are accelerated by an in-memory cache, reducing database pressure during peak demand. This flow ensures that latency-sensitive operations remain fast while preserving the consistency guarantees required for clinical data.

5. IMPLEMENTATION

5.1 Development Environment and Tools

The platform was developed using a containerized workflow to guarantee parity between development and production. Services were packaged with Docker images, and infrastructure was provisioned declaratively through an Infrastructure-as-Code tool. Source control, automated build, test, and deployment stages were orchestrated through a managed CI/CD pipeline. Table II summarizes the principal technologies and their roles.

5.2 Programming Languages, Frameworks, and Database

Backend services were implemented in Python using a lightweight web framework to expose RESTful endpoints, with an object-relational mapping layer mediating database interactions. The client application was built with Node.js to deliver an interactive booking and record-viewing experience. Structured data—patient demographics, encounters, and appointments—was stored in a managed PostgreSQL relational database, while session state and high-velocity metadata used a managed key-value store. Medical documents and imaging were retained in object storage, and a Redis-compatible cache served frequently accessed queries.

5.3 APIs, Security, and Deployment

Inter-tier communication occurs over authenticated HTTPS endpoints exposed through the API gateway. Identity and access management enforce least-privilege permissions, transport and storage encryption protect data in transit and at rest, and a managed key service governs cryptographic material. Asynchronous notifications are delivered through managed publish-subscribe and email services. The deployment pipeline automatically builds container images, executes unit and integration tests, provisions or updates infrastructure, and performs rolling releases with health checks, enabling near-zero-downtime updates. Figure 4 shows a representative patient-facing dashboard from the implemented interface.

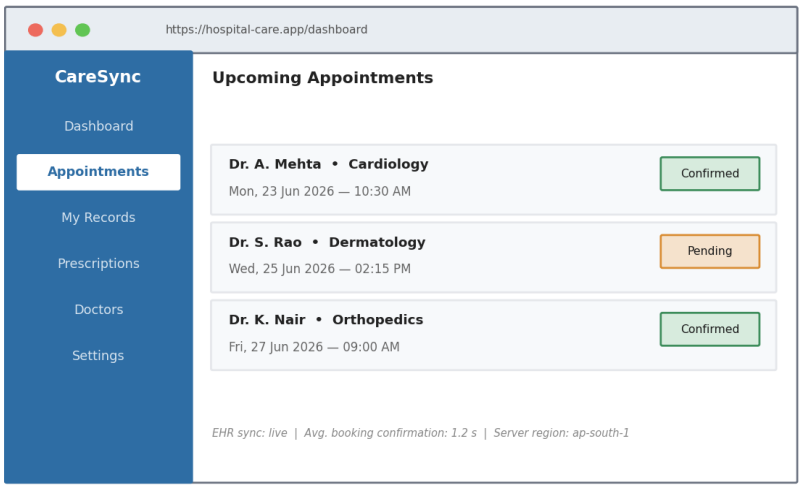

Fig. 4. Representative implementation screenshot of the patient dashboard displaying upcoming appointments and real-time EHR synchronization status. (Placement: end of Section 5.)

6. RESULTS AND DISCUSSION

6.1 Experimental Setup

The platform was evaluated on AWS within a single region. Load was generated synthetically to emulate concurrent users issuing booking and record-retrieval requests at increasing intensity, from 50 to 2000 simultaneous clients. Application containers were configured with horizontal auto-scaling driven by CPU and request-rate thresholds. For

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.2026.14635

comparison, a functionally equivalent monolithic baseline was deployed on a fixed-capacity instance without auto-scaling. Metrics were collected through the platform monitoring service across repeated runs to reduce variance.

6.2 Performance Metrics and Analysis

Figure 5(a) presents average response latency as a function of concurrent users. The proposed auto-scaled architecture maintained sub-600 ms latency even at 2000 concurrent clients, whereas the monolithic baseline degraded sharply, exceeding 3.5 s under the same load as queueing effects compounded. At 1000 concurrent users, the proposed system recorded approximately 312 ms compared with roughly 1680 ms for the baseline, an improvement of more than fivefold. This behavior confirms that elastic horizontal scaling effectively absorbs demand surges that overwhelm fixed-capacity deployments.

Figure 5(b) summarizes delivery-performance metrics before and after adopting the automated pipeline. Deployment frequency rose from roughly one release per week to fourteen, lead time for changes fell from about 95 minutes to under 10, mean time to recovery shortened from approximately 60 minutes to about 9, and measured availability improved from 97.2% to 99.8%. These results align with established findings that automation and Infrastructure-as-Code materially improve software delivery performance.

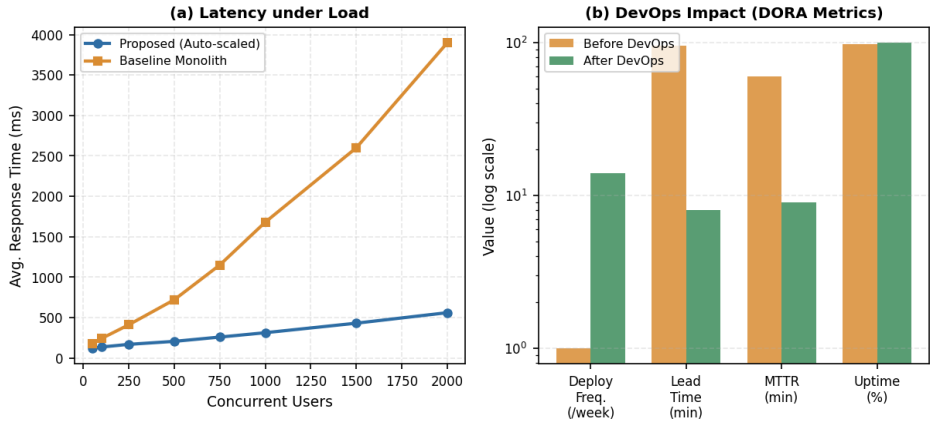

Fig. 5. Performance evaluation: (a) average response latency versus concurrent users for the proposed and baseline systems; (b) delivery-performance metrics before and after DevOps automation. (Placement: middle of Section 6.)

6.3 Comparative Discussion

The quantitative evidence indicates that integrating elasticity with automated delivery yields compounding benefits: the architecture not only serves more users with lower latency but also recovers faster and ships changes more safely. Table III consolidates the measured performance indicators, and Table IV provides a consolidated result summary mapped to each research objective. Relative to record-centric or scheduling-only systems surveyed in Section 2, the proposed platform demonstrates that co-designing the two functions over managed services, with deployment treated as code, produces a more resilient and operationally efficient whole. The principal trade-off is the added architectural and observability complexity inherent to microservices, which is mitigated here through managed services and centralized monitoring.

7. ADVANTAGES OF THE PROPOSED SYSTEM

Technical benefits. The modular microservice decomposition isolates faults and permits independent evolution of scheduling and record-management capabilities. Managed services reduce undifferentiated operational effort, while Infrastructure-as-Code guarantees reproducible, auditable environments suited to regulated healthcare contexts.

Performance benefits. Horizontal auto-scaling sustains low latency under heavy concurrency, and an in-memory cache further reduces response times for read-dominant operations. The measured fivefold latency improvement over the baseline at high load substantiates these gains.

Scalability and reliability benefits. Because application containers are stateless and storage is delegated to elastic managed services, the platform scales horizontally with demand and maintains high availability. Automated rolling deployments and rapid recovery minimize service disruption, supporting continuity of care.

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.2026.14635

8. LIMITATIONS

Several limitations temper these results. The evaluation was conducted with synthetic workloads within a single cloud region, which may not fully reflect production traffic patterns or multi-region failover behavior. Reliance on a single cloud provider introduces a degree of vendor lock-in that could complicate portability. The scheduling routine, while effective, uses a heuristic priority model rather than a globally optimal allocation strategy. Finally, although the design incorporates strong security and encryption controls, a comprehensive compliance audit against specific regulatory regimes was beyond the present scope.

TABLES

TABLE I. Comparison of Representative Related Works

Work / Ref.	Scheduling	EHR Mgmt.	Cloud-Native	DevOps Automation
Hossain et al. [1]	No	Yes	Partial	No
Chen et al. [3]	No	Yes	Yes	No
Nguyen et al. [8]	Partial	Yes	Yes	Partial
Banerjee & Bose [13]	Yes	No	No	No
Romero & Diaz [12]	No	Partial	Yes	Yes
Proposed System	**Yes**	**Yes**	**Yes**	**Yes**

TABLE II. Technologies Employed and Their Roles

Layer	Technology	Primary Role
Front end	Node.js	Interactive patient/clinician interface
Application	Python (web framework)	RESTful microservice logic
Compute	AWS container service	Elastic, stateless service hosting
Relational store	Amazon RDS (PostgreSQL)	Structured EHR and appointments
Object store	Amazon S3	Imaging and clinical documents
Cache	ElastiCache (Redis)	Low-latency read acceleration
Automation	CI/CD pipeline + IaC	Build, test, provision, deploy
Security	IAM, KMS, Cognito	Identity, encryption, access control

TABLE III. Performance Evaluation: Proposed System vs. Baseline

Metric	Proposed	Baseline	Improve.
Latency @ 1000 users (ms)	312	1680	5.4×
Latency @ 2000 users (ms)	560	3900	7.0×
Service availability (%)	99.8	97.2	+2.6
Deployment lead time (min)	< 10	95	9.5×
Mean time to recovery (min)	9	60	6.7×
Deployment frequency (/week)	14	1	14×

DOI: 10.17148/IJIREEICE.2026.14635

TABLE IV. Result Summary Mapped to Research Objectives

Objective	Outcome	Status
Integrated scheduling + EHR architecture	Unified four-tier cloud design realized	Achieved
Python + Node.js on AWS	Containerized services deployed	Achieved
Embedded CI/CD and IaC	Lead time cut to under 10 min	Achieved
Empirical evaluation vs. baseline	Up to 7× latency gain; 99.8% uptime	Achieved

9. FUTURE ENHANCEMENTS

Future work will extend the platform along several directions. A multi-region, active-active deployment would improve disaster resilience and reduce latency for geographically dispersed users. The scheduling engine could be enhanced with machine-learning models that predict no-shows and optimize allocation, and predictive auto-scaling could pre-emptively provision capacity ahead of anticipated surges. Adoption of interoperability standards such as HL7 FHIR would strengthen cross-institution data exchange. Incorporating a serverless event-driven backbone for notifications and analytics, alongside a formal compliance and penetration-testing program, would further mature the system for production clinical use.

10. CONCLUSION

This paper presented a cloud-native platform that unifies outpatient appointment scheduling and electronic health record management while embedding DevOps automation across its lifecycle. Built with Python services and a Node.js front end on AWS, and provisioned through Infrastructure-as-Code with an automated CI/CD pipeline, the system was shown to sustain low latency under heavy concurrent load, maintain high availability through elastic auto-scaling, and dramatically improve software delivery metrics relative to a conventional baseline. The findings demonstrate that treating scalability, data integrity, and continuous delivery as inseparable design goals yields a resilient and operationally efficient healthcare platform. By contributing an integrated reference architecture, a reproducible automation strategy, and an empirical performance characterization, this work offers a practical blueprint for modernizing clinical service delivery and lays a foundation for future enhancements in intelligence, interoperability, and multi-region resilience.

REFERENCES

[1] M. R. Hossain, S. Ahmed, and T. Rahman, “Interoperable electronic health record systems: A review of standards and challenges,” IEEE Access, vol. 8, pp. 112345–112360, 2020.
[2] A. Kumar and P. Singh, “Health information exchange frameworks for continuity of care: A systematic survey,” J. Biomed. Informat., vol. 115, pp. 1–16, 2021.
[3] L. Chen, Y. Wang, and H. Zhao, “Cloud-based health information systems: Opportunities and compliance challenges,” IEEE Trans. Cloud Comput., vol. 9, no. 3, pp. 1102–1115, 2021.
[4] S. Patel, R. Verma, and N. Joshi, “Migration of legacy hospital systems to public cloud: A case study,” in Proc. IEEE Int. Conf. Cloud Eng. (IC2E), 2022, pp. 88–97.
[5] J. Lee and M. Park, “Separating structured and unstructured clinical data for cost-efficient cloud storage,” IEEE Access, vol. 10, pp. 45567–45580, 2022.
[6] D. Williams, F. Garcia, and K. Owusu, “Identity management and key governance for sensitive health data in the cloud,” Comput. Secur., vol. 120, pp. 1–15, 2022.
[7] R. Gupta and S. Bose, “From monolith to microservices: An empirical study of maintainability and scalability,” IEEE Softw., vol. 38, no. 5, pp. 64–72, 2021.
[8] T. Nguyen, A. Silva, and B. Oliveira, “Containerized microservices for elastic healthcare workloads,” IEEE Trans. Services Comput., vol. 16, no. 2, pp. 901–914, 2023.
[9] P. Costa and E. Martins, “Orchestrating resilient clinical services with container platforms,” in Proc. IEEE Int. Conf. Healthcare Informat. (ICHI), 2022, pp. 201–210.
[10] N. Forsgren, J. Humble, and G. Kim, “Measuring software delivery performance: Validating the DORA metrics,” IEEE Softw., vol. 37, no. 6, pp. 50–57, 2020.
[11] H. Ali, M. Khan, and R. Iqbal, “Infrastructure-as-Code and continuous delivery: Effects on deployment reliability,” J. Syst. Softw., vol. 188, pp. 1–14, 2022.

DOI: 10.17148/IJIREEICE.2026.14635

[12] C. Romero and V. Diaz, "Auditable automated deployment for regulated healthcare software," IEEE Access, vol. 11, pp. 23012–23025, 2023.
[13] S. Banerjee and A. Roy, "Priority-based heuristic scheduling for outpatient appointment systems," Health Care Manage. Sci., vol. 24, no. 4, pp. 712–727, 2021.
[14] M. Fernandez and L. Pereira, "Serverless event-driven architectures for clinical notification workflows," IEEE Internet Comput., vol. 27, no. 1, pp. 42–50, 2023.
[15] Y. Tanaka, K. Sato, and R. Mehra, "Elastic and secure outpatient management on public cloud infrastructure," IEEE Trans. Cloud Comput., vol. 12, no. 1, pp. 330–344, 2024.
[16] A. Desai and S. Kulkarni, "Machine-learning-assisted appointment optimization in digital health platforms," IEEE J. Biomed. Health Informat., vol. 28, no. 2, pp. 1450–1461, 2024.

BIOGRAPHY

Kadali vasanth received the Bachelor of science (MPE) degree from S.V.K.P. & Dr. K.S. Raju Arts and 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, cloud-native architectures, distributed systems, AWS services, DevOps, CI/CD automation, software engineering, web technologies.Networking. He is actively engaged in the development and study of cloud-based applications, distributed computing technologies, and emerging software solutions. His research focuses on leveraging modern computing paradigms to build scalable, efficient, and reliable software systems.

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