A Unified Experimental Framework for Evaluating and Optimizing Cloud Deployment Architectures in Smart Healthcare Systems

Norhan Abdel-Hamid[1], Mohamed Sabry Fouad Saraya[2], Amr M.T. Ali-Eldin[3], John Zaki[4]

[1]Computer Eng. and Control sys Dept. Faculty of Engineering, Mansoura University, Egypt
Norhan.abdelhamid@mans.edu.eg
[2]Associate professor, Computers and control systems department, Mansoura University faculty of engineering
01000741524, Mansoura – Egypt
mohamedsabry83@mans.edu.eg
[3]Computer Engineering and Control System Dept., Mansoura university
amr.ali-eldin@mans.edu.eg
[4]Department of computer and Control system Dept., Mansoura university
Jfzaki@mans.edu.eg
Corresponding author: Norhan Abdel-hamid, Norhan.abdelhamid@mans.edu.eg

HOW TO CITE:
Norhan Abdel-Hamid, Mohamed Sabry Fouad Saraya, Amr M.T. Ali-Eldin, John Zaki (2026). A Unified Experimental Framework for Evaluating and Optimizing Cloud Deployment Architectures in Smart Healthcare Systems. International Journal of Special Education, 41(5s), 498-530.

ABSTRACT:

In this paper, we have proposed a detailed evaluation of the deployment approaches of smart healthcare systems in the cloud computing environment, with particular emphasis on the performance comparison of Service-Oriented Architecture (SOA), containerized microservices on Amazon EC2, and serverless microservices on AWS Lambda. The proposed framework has incorporated real-time healthcare services, including patient registration, consultation, and notification, under a realistic workload model, simulating mixed read/write operations with bursty traffic patterns. The authors have also proposed the empirical evaluation of the proposed framework using a controlled experimental setup, including Linux-based EC2 instances (t3. medium), containerized Flask-based microservices, and event-driven Lambda-based microservices with concurrency limits on the number of requests. The authors have also collected the performance metrics of the proposed framework, including the average response time, throughput, CPU utilization, dropped requests, and cost per 10,000 requests. The authors have also demonstrated the performance evaluation of the proposed framework, where the serverless-based microservices (Lambda) have achieved the least average response time of 0.98 seconds, outperforming the performance of EC2-based microservices (1.05 seconds) and SOA (1.45 seconds). The authors have also demonstrated the performance evaluation of the proposed framework, where the serverless-based microservices (Lambda)

have achieved the highest scalability of 2400 requests per second, outperforming the performance of EC2-based microservices (1400 requests per second) and SOA (900 requests per second). Moreover, Lambda exhibits better reliability with a lower rate of dropped requests (150), whereas SOA suffers a high degree of performance degradation with a high rate of dropped requests (1600). Cost analysis also revealed that the proposed approach using AWS Lambda is the most cost-effective approach with a cost of 0.65 USD for every 10k requests, compared to EC2 (0.85 USD) and SOA (1.20 USD). Moreover, a qualitative evaluation using expert judgment with a panel of 25 experts also supported that microservices architectures exhibit better performance in terms of increased resilience, agility in testing, and maintainability. From the analysis, it is evident that serverless microservices architectures exhibit a high degree of benefits for dynamic event-driven healthcare systems, whereas a hybrid approach using EC2 and Lambda balances both steady-state and bursty workloads for healthcare cloud systems. The study also provides a holistic approach for evaluating cloud architectures using performance metrics with decision-making for modern healthcare cloud systems.

Keywords: Microservices; Smart healthcare; Cloud computing; Wearable devices; Internet of Things (IoT); Service-based architecture (SOA)

COPYRIGHT STATEMENT:
Copyright: © 2026 Authors. Open access publication underthe terms and conditions of the Creative CommonsAttribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

A. Motivation

The rapid digitalization of healthcare systems, facilitated by the widespread adoption of IoT-enabled medical devices and applications of continuous patient monitoring, has significantly altered the operational demands of modern healthcare infrastructures. Unlike traditional healthcare information systems, which tend to be static and transaction-based, contemporary smart healthcare systems are naturally dynamic, distributed, and event-based with continuous data flows and time-critical events such as anomaly detection and alerts, and real-time clinical decision support systems. This new paradigm places stringent demands on system architectures to ensure not only scale and availability but also low latency, elasticity, fault isolation, and cost efficiency under varying and potentially explosive workloads. To address these stringent demands on system architectures for smart healthcare systems, recent research has increasingly advocated for cloud-native paradigms such as microservices and serverless computing. These two paradigms have been identified as potential architectural models for smart healthcare systems. Microservices provide modularity and flexibility in deployment and fault isolation. Serverless computing offers fine-grained elasticity and pay-per-use cost models.

However, despite this increasing interest, the existing body of work still appears to be fragmented and inconclusive from the point of deployment decision-making. Most existing work has been focused on a specific architectural paradigm or has only considered a limited set of factors such as scalability and security, without providing a unified and experimentally supported evaluation of different deployment options under various healthcare workloads. As shown in Table 1, none of the existing work has been able to provide a comprehensive evaluation for architectural comparison, deployment strategies, dynamic workload handling, and multi-criteria evaluation altogether. This has resulted in a crucial gap between theoretical architectural suitability and deployment decision-making. Healthcare system design practitioners are still unaware of the selection

criteria for the best deployment option under mixed workload conditions where both continuous and burst-driven workloads are present. In particular, it is still unknown how the suitability of traditional service-oriented architectures, VM-based microservices (e.g., EC2), and serverless execution environments (e.g., Lambda) would compare under the same healthcare system design requirements in terms of performance metrics such as latency, throughput, scalability, cost, and other related factors. Moreover, the evaluation of hybrid deployment strategies, where the benefits of both persistent and event-driven execution can be maximized, has yet to be undertaken.

Keeping the aforementioned challenges and gaps in the existing body of work in mind, the present work aims to move beyond the existing fragmented architectural design and evaluation for smart healthcare systems. By modeling healthcare workloads for both continuous and burst-driven scenarios and providing a comprehensive evaluation for various deployment options under a unified set of performance and other related metrics, the proposed work aims to bridge the gap between architectural design and deployment strategy for healthcare system design. In this regard, the proposed work not only adds to the existing body of work but also provides insights that can be potentially useful for the design of more efficient, scalable, and adaptive healthcare systems for the future.

B. Literature review

The increased transformation of the healthcare industry into a more connected, data-centric, and continuously monitored environment has created a growing need for software architectures that are not just rich in functionality, but also scalable, resilient, secure, and responsive to dynamic workloads. In this sense, the literature is converging around cloud-native healthcare architectures, IoT-based medical monitoring, wearables, microservices-based system design, among other areas, as enablers for the development of smart healthcare systems. Therefore, despite the evident success in advancing the development of smart healthcare architectures, the literature still lacks a unified evaluation of the suitability of different architectures for the healthcare domain, with many studies focusing on either the design of the architecture, security considerations, or the development of a particular aspect of the system, with fewer rigorous evaluations of different architectures under a unified set of healthcare workloads. This is particularly important for the purpose of the current research, which is not just a proposal for another healthcare architecture, but rather a proposal for the evaluation of the suitability of different architectures under the realities of smart healthcare systems. In this sense, the proposed literature classification and gap mapping matrix in Table 1 provides a useful basis for understanding the literature on smart healthcare architectures, with the heatmap providing a better visualization of the coverage of evaluation dimensions in the literature.

A first major stream is related to the design of healthcare platforms and the use of ecosystem thinking, where the focus is on the structuring of services and the relationship between stakeholders, as well as the creation of a digital healthcare environment. In this direction, Ruokolainen et al. [1] proposed the design of digital healthcare ecosystems for the management of Parkinson's disease. This study clearly indicated the need for healthcare platforms to enable the interaction of multiple actors, the integration of services, and the creation of value for the patient. Moreover, the proposed framework for healthcare platforms in [2] introduced a cloud-assisted model for the development of software applications through a microservice-based approach. This study indicated the advantages related to the use of modular structures and the decomposition of services. Other related works, such as [3] and [4] further expanded this direction by introducing cloud-native middleware and e-health systems for the elderly. In addition, the digital transformation needs for healthcare organizations were discussed in [5] This body of work clearly indicated the importance of modularity and the decomposition of services for healthcare platforms. However, the main contribution of this work is related to the conceptual part. As may be seen in Table 1, these studies generally fall on the conceptual side of the evaluation spectrum and do not follow through to performance evaluation of the deployment

alternatives considered. This is also evident in the heatmap, where these studies have the least coverage of the dimensions of latency, throughput, cost, and deployment comparison. Therefore, although these studies offer a compelling case for the architecture of cloud-native healthcare systems, they do not adequately address the question of which deployment model might be more suitable in the real world, given the demands of healthcare systems operation.

A second, highly relevant body of work has examined the topic of IoT-enabled healthcare systems, wearable technologies, and telemonitoring applications. This body of work is particularly significant because smart healthcare environments are increasingly enabled by continuous sensing, remote monitoring, and event-driven clinical workflows. Walle et al. [6] investigated the intention to use wearable health devices, indicating that this is influenced by both technological and user-centric factors, which are significant to healthcare deployment scenarios, although the work does not examine architecture per se. On a higher-level system basis, the telemonitoring framework proposed by [7], as well as the broker less IoHT platform proposed by [8], showed the feasibility of microservice-based systems in supporting distributed monitoring and intelligent handling of healthcare information. The work by Subhan et al. [9] surveyed the feasibility of AI-based wearable medical Internet of Things systems and emphasized the importance of intelligent analytics, real-time processing, and scalable digital care. Other research works such as [10], [11], [12] further emphasized the importance of service decomposition in handling sensor-based and predictive healthcare information. The strength of this group of research works is their close association and relevance to the dynamics of healthcare, particularly continuous monitoring and asynchronous event generation. The weakness of this group of research works is that they mainly focus on a single approach, such as microservices, without comparing it to other possible approaches such as SOA deployment or serverless execution. In conclusion, it is evident from this group of research works that microservice-based healthcare services are suitable and feasible for Internet of Things-based care systems. However, it is less evident and less conclusive when the research works are applied to a broader context such as deployment efficiency.

The third group of research works deals with cloud deployment models, scalability models, and microservice execution models. The work by Sadek et al. [13] emphasized the importance of microservices and serverless computing models in healthcare environments. The work by Jangra and Mangla [14] proposed intelligent load balancing and scheduling models for healthcare cloud systems. Aksakalli et al. [15] offered a systematic method for developing feasible deployment alternatives for microservices, which is particularly relevant from a decision support point of view, even though the original work is not health-related. Among the most impactful works within this cluster, Calderón-Gómez et al. [16] compared service-oriented and microservice architecture patterns for deploying eHealth applications in cloud environments, which is particularly impactful for this research. Furthermore, [17] studied dynamic deployment for local edge computing environments, [18] studied serverless microservice architecture for cloud-edge intelligence, and [19] offered a review for smart healthcare in serverless environments. Overall, this cluster of works clearly demonstrates that deployment architecture is not a secondary aspect for systems, but rather a primary factor that affects overall system performance. However, the limitations of this body of work are substantial. While [16], is an impactful work that is particularly relevant for this research, it does not extend the comparison to serverless environments. Furthermore, [18] provides insights for serverless environments, but without a proper point of comparison for serverless environments, namely EC2-based microservices and SOA. Similarly, [19] provides a review for serverless environments, which is again highly conceptual. As demonstrated in Table 1, this body of work provides a fragment of the overall deployment space, but not the complete picture. This is also clearly demonstrated in the heatmap, where different works provide a fragment of the overall deployment space, but no work provides a complete solution for the deployment space, which covers scalability, dynamic workloads, and architectural comparison.

Another significant research stream deals with interoperability, privacy, security, and compliance. This research stream is critical in the healthcare domain, as architectural superiority can never be divorced from reliability. A reference architecture for a patient-centric system, using blockchain, cloud, and IoT for semantic interoperability, was proposed by Gohar et al. [20] Other research, such as [21][22] discussed the interoperability of standardized electronic health records, and [23] discussed secure and HIPAA-compliant microservices for healthcare data access and prescription processing. Zala et al. [24] discussed secure and privacy-preserving patient record management on third-party cloud platforms, and [25] continued this research direction using blockchain-based federated microservices. This research is extremely valuable because it reveals non-negotiable architectural requirements for healthcare systems, especially when it comes to issues of confidentiality, trust, and interoperability. However, it is more governance-centric rather than performance-centric. We learn from this research that healthcare services should be secure and interoperable, but we do not learn much about how different systems behave in terms of latency, throughput, scalability, or cost. Therefore, they are an essential part of the context for healthcare architecture design but do not obviate the need for the evaluation of the deployment.

A further body of work in the literature relates to the topic of resilience, fault tolerance, adaptive healthcare systems, and edge intelligence. do Nascimento et al. [26] presented a discussion on the topic of self-adaptive IoT architecture for e-health systems and found the need for healthcare environments to have systems that can react to the variability present during execution. A further work [27] proposed a scheme for a microservices-enabled bidirectional fault-tolerant approach for Healthcare IoT systems. In the field of AI, [28] and [29] discussed medical cyber-physical systems and real-time health-related data processing. On the AI side, [30] proposed a distributed edge-to-cloud deep learning architecture for healthcare diagnosis. This body of work is particularly relevant to the topic as it gets closer to the realities of the healthcare environment by addressing faults, adaptability, and distributed intelligence. Nevertheless, the body of work still appears to be focused on enhancing one architectural path rather than comparing the options under a unified evaluation criterion. This means the work is useful for refining the objectives for the design but does not assist in the selection of the architecture from a variety of deployment options.

C. Research gap

An overview of the entire body of literature reveals the existence of certain trends. First and foremost, the consensus among the majority of the literature is the suitability of modular, distributed, and cloud-enabled architectures for the ecosystem-oriented approach to the design of smart healthcare systems over monolithic systems. This is supported by ecosystem-oriented literature such as [1] and [2], IoT healthcare-oriented literature such as [7] and [8], and deployment-oriented literature such as [13], [31], and [16]. Secondly, the literature is consistent in proving the point that the requirements for the deployment of a smart healthcare system are not only performance-based but also include security, privacy, and resilience requirements, as supported by [20], [22], [23], [24], [25], [27]. Thirdly and most importantly, the body of literature is fragmented in the evaluation approach. While some literature is more inclined towards the conceptual approach, some is more inclined towards the domain approach, while others are more inclined towards the edge logic approach and the security approach. This is the point where Table 1 comes into play. The table does not only present the literature but also maps the literature according to the architecture type, deployment model, workload type, evaluation level, main limitation, and the gap. This reveals the fact that the main limitations are the lack of multi-architecture comparison, the lack of workload benchmarking, and the lack of unified performance metrics. The heatmap also supports this conclusion visually by demonstrating sparse and unbalanced results in critical areas such as architecture comparison, deployment comparison, latency, throughput, cost, and completeness of experiments. As a result of this synthesis of related works, the research gap identified by this study can be clearly articulated. The current body of knowledge does

not sufficiently offer a controlled and experimentally validated comparison of traditional service-oriented deployment models, EC2-based microservices, and serverless microservices for delivering a given healthcare-related application scenario. Although works such as [16] attempt to address this gap to a degree, they fail to do so by considering only SOA and microservices deployment models and ignoring serverless computing. Other works such as [18] and [19] while recognizing the potential of serverless computing for microservice deployment models, fail to compare serverless computing with server-based deployment models under similar conditions. Other works such as [7], [8], [27]while demonstrating practical implementations of different deployment models in healthcare-related scenarios, fail to do so by considering only a single deployment model. As a result of this gap in knowledge, there still remains an important research question to be addressed by a comprehensive solution: what deployment model offers the best trade-off between responsiveness, scalability, elasticity, reliability, and feasibility for smart healthcare-related applications with both continuous and event-driven behavior? This is the precise gap that the present study addresses.

D. research contribution

In this regard, the proposed work should be viewed not as a standalone healthcare-related application for a microservice-based system, but rather as a comparative and decision-oriented work. This is so because, while previous works were limited to a single paradigm and/or a limited set of metrics for evaluation, the proposed model assesses the three deployment options under a unified approach: the traditional SOA-based deployment option, the EC2-based microservices option, the Lambda/serverless option, and the relevance of the hybrid deployment behavior. This is necessary because smart healthcare workloads are not usually static or even purely event-driven; rather, workloads often combine the continuous flow of information from the environment with sudden peaks in activity driven by abnormal events, alerts, and other context-specific escalations. In this regard, a fair assessment of the architecture should not only consider theoretical claims but also test the behavior of each deployment option under mixed workload conditions. In this regard, the proposed work advances the body of literature from a descriptive approach advocating the use of cloud-native healthcare architectures to an evidence-based approach to evaluating the architecture. The contribution of the proposed work can be viewed at two levels: the conceptual contribution and the contribution to the body of literature. At the methodological level, it creates a unified comparative context in which different deployment paradigms can be comparatively analyzed according to a unified set of criteria, rather than different criteria for different works or even different criteria for different sections of a given work. At the analytical level, it overcomes a literature gap between healthcare architecture-oriented research and cloud-based performance-oriented research on service deployment. At the practical level, it can be useful for healthcare system designers who are tasked with making a choice between stable VM-based microservices, highly elastic serverless services, or even combinations of these, depending on the nature of the workload and other operational considerations. Above all, as indicated in the final row of Table 1 and as indicated in the full coverage profile in the heatmap, the proposed work differs from existing works in that it offers a more comprehensive evaluation context in which architecture comparison, deployment strategy, dynamic workload behavior, and experimental validation are considered simultaneously, thus positioning the proposed work in a stronger position relative to the existing literature and providing a clearer and more defensible research contribution.

Overall, the literature provides a strong justification for the need for cloud-native and microservice-based healthcare systems; however, it does not fully address the question of relative deployment that is so crucial for real-world systems. By addressing this issue directly through a unified framework for comparison, this work aims to provide a more rigorous and relevant contribution to the body of literature on smart healthcare architectures than what is currently available. Rather than simply reiterating that microservices or serverless computing is a good approach for smart healthcare

architectures, this work aims to determine under what circumstances one approach is better than another, thereby providing a more in-depth contribution than the majority of the literature that currently exists on the topic. In order to address the research gap that has been identified, the rest of this paper is divided into sections that progressively develop and validate the proposed evaluation framework. First, the overall system architecture and the methodology that was adopted for developing the proposed framework for modeling smart healthcare workload scenarios are described. Following this, the actual implementation of the proposed architectural paradigms is described, along with the experimental setup that was adopted for comparing the relative benefits of each approach. Finally, a detailed analysis of the results that were obtained is provided. Finally, the findings are discussed in relation to the existing body of literature, including key insights, applications, and avenues for further research.

Table 1. Integrated Literature Classification and Gap Mapping for Smart Healthcare Architectures

Ref	Category	Architecture	Deployment Model	Workload Type	Evaluation Level	Key Contribution	Main Limitation	Identified Gap
[1]	Platform	Ecosystem	Conceptual Cloud	Static	Conceptual	Healthcare ecosystem design	No experimental validation	No deployment comparison
[13]	Deployment	Microservices / Serverless	Cloud	General	Conceptual	Highlights scalability potential	No quantitative evaluation	Lacks controlled experiments
[6]	Wearable	IoT-based	Edge/Cloud	Dynamic	User-level	Adoption analysis	No system architecture evaluation	No performance metrics
[14]	Deployment	Cloud-based	Cloud	Moderate	Experimental	Load balancing optimization	Focus on scheduling only	No architecture comparison
[20]	Security	Blockchain + Cloud	Hybrid	Static	Conceptual	Interoperability framework	No runtime validation	No performance analysis
[5]	Platform	Digital Healthcare	Cloud	Static	Conceptual	Digital transformation insights	No system implementation	No deployment evaluation
[21]	Interoperability	EHR Systems	Cloud	Static	Conceptual	Standardization	No system validation	No architecture impact study
[22]	Security	Microservices	Cloud	Moderate	Experimental	HIPAA-compliant architecture	Limited scope	No multi-architecture comparison
[23]	Security	Microservices	Cloud	Moderate	Experimental	Secure data access	Narrow focus	No scalability evaluation
[2]	Platform	Microservices	Cloud	Static	Conceptual	Dev framework	No runtime analysis	No deployment

								comparison
[3]	Platform	Middleware	Cloud	Static	Conceptual	Cloud-native healthcare middleware	No experiments	No workload modeling
[17]	Edge	Microservices	Edge-Cloud	Dynamic	Experimental	Dynamic deployment	Partial evaluation	No baseline comparison
[18]	Serverless	Microservices	Edge-Cloud	Dynamic/Burst	Experimental	Serverless edge intelligence	Limited metrics	No comparison with EC2/SOA
[15]	Deployment	Microservices	Cloud	General	Experimental	Deployment alternatives	Not healthcare-specific	No workload-driven evaluation
[7]	IoT	Microservices	Cloud	Dynamic	Experimental	Telemonitoring system	Single architecture	No deployment comparison
[8]	IoT	Microservices	Brokerless Cloud	Dynamic	Experimental	IoHT architecture	No benchmarking	No architecture comparison
[16]	Deployment	SOA vs Microservices	Cloud	Static	Experimental	Architecture comparison	Limited scope	No serverless evaluation
[9]	Survey	IoMT + AI	General	Dynamic	Review	Comprehensive survey	No experiments	No quantitative comparison
[10]	IoT	Microservices	Cloud	Dynamic	Experimental	Distributed monitoring	Non-human domain	Limited generalization
[19]	Serverless	Healthcare	Cloud	Dynamic	Conceptual	Serverless review	No implementation	No experimental validation
[11]	AI + IoT	Microservices	Cloud	Dynamic	Experimental	Disease prediction	Narrow scope	No architecture comparison
[28]	CPS	Healthcare Systems	Hybrid	Dynamic	Conceptual	Cyber-physical integration	No validation	No performance evaluation
[27]	Resilience	Microservices	Cloud	Dynamic	Experimental	Fault tolerance	Single architecture	No deployment comparison

[12]	IoT	Healthcare	Cloud	Dynamic	Conceptual	IoT integration	No implementation	No evaluation
[30]	AI	Edge-Cloud	Hybrid	Dynamic	Experimental	Edge AI diagnosis	AI-focused	No deployment analysis
[25]	Security	Blockchain + Microservices	Hybrid	Static	Conceptual	Federated privacy	No runtime eval	No scalability analysis
[4]	Platform	e-Health	Cloud	Static	Conceptual	Elderly healthcare system	No evaluation	No deployment comparison
[31]	IoT	Microservices + Serverless	Hybrid	Dynamic	Experimental	Secure IoT architecture	Not healthcare-specific	Limited healthcare validation
[32]	Monitoring	Microservices	Cloud	Dynamic	Experimental	Scalable monitoring	Not healthcare-specific	No architecture comparison
This Work	Comparative	SOA + EC2 + Lambda	Hybrid	Dynamic + Burst	Experimental	Full architecture comparison		

2. Proposed driven framework design thinking

Design thinking is a human-centered and iterative problem-solving strategy aimed at understanding user needs and requirements while challenging assumptions and redefining complex problems to facilitate innovative and practical solution development. This paradigm has been successfully incorporated by leading technology companies like Apple, Google, and IBM to facilitate innovation in their products and improve user-centric system development. The process of design thinking involves five sequential phases: Empathize, Define, Ideate, Prototype, and Implement. Each phase of the process contributes to improving the solution by facilitating continuous user interactions. Thus, considering the user requirements of healthcare systems to be complex and context-dependent, the design thinking paradigm is most suitable for developing an effective framework for Smart Healthcare based Microservice Application (SHbMSA). As shown in Figure 1, the proposed framework would transform user requirements into deployable microservice applications by incorporating a structured framework. The framework would be divided into two main layers: Information Layer and Integration Layer. These two layers would be responsible for bridging the gap between user requirements and system implementation.

The Information Layer is the first layer in the system development process, which deals with the initial phase of system development, where raw requirements are gathered, analyzed, and structured. This layer is analogous to the Empathize and Define phases of design thinking, where the needs of the stakeholders are understood, and system requirements are formally defined. In this layer, requirements are gathered from different sources, including patients, clinicians, IoT devices, healthcare systems, regulatory factors, standards, and rules, which are stored in a centralized repository for traceability and consistency throughout the system development process. Then, the gathered requirements are structured, where they are categorized as functional and non-functional requirements, and are grouped based on domain-specific contexts. This enables the system

architects to clearly identify service boundaries and prepare for microservice decomposition. Additionally, several architectural factors are considered for the development of robust and scalable systems, which include: (1) Identifying the business capabilities with the help of domain experts. (2) Defining clear boundaries for microservices, which provide loose coupling and high cohesion between services. (3) Defining service communication contracts, which include APIs, data formats, and protocols. (4) Dealing with non-functional requirements, which include security, monitoring, logging, and fault tolerance. (5) Incorporating feedback from different stakeholders, which is considered during the elicitation of requirements. (6) Dealing with non-functional requirements, which include scalability, performance, and maintainability. The Information Layer is a structured layer that transforms unorganized demands from different stakeholders into clear system specifications.

The Integration Layer is a representation of the transformation phase, where structured requirements are transformed into fine-grained, event-driven microservices. This layer is a reflection of the Ideate, Prototype, and Implement phases of design thinking, where solutions are ideated, developed, and implemented, respectively. In this phase, service decomposition occurs, where business domains are transformed into independent microservices, each designed to deliver a particular capability. This is done based on functional and non-functional requirements defined in the previous layer. Here, an event-driven architecture is used, where microservices use asynchronous messaging mechanisms for interactions, thus enhancing the responsiveness of the system, which is critical for real-time healthcare data processing. This layer is divided into several sub-phases, namely: (1) Ideation Phase: This is where candidate microservices are identified, where different architectural options are explored, and where PoC prototypes are developed. (2) Development Phase: This is where microservices are developed, coded, and where system components are tested and validated. (3) Implementation Phase: This is where microservices are deployed in a cloud environment, where scalability, availability, and reliability are ensured.

Support and Feedback Phase: Continuous monitoring, feedback collection, and refinement of the system Although the system comprises independent microservices, the entire system works in a cohesive manner within the healthcare system as a whole. One of the significant features of the proposed framework is the feedback loop, as depicted in Figure 1 above. This framework is not a linear process; rather, the entire process continuously evolves with performance evaluation, feedback, and refinement of the entire system. This feedback process ensures: (1) Continuous refinement of the system. (2) Continuous optimization of the system based on performance evaluation. (3) Continuous adaptation of the system according to changing healthcare needs.

Although the conceptual framework proposed in this section depicts the entire methodology, it acts as a theoretical framework for the entire methodology proposed in this paper. Although Figure 1 above depicts the entire transformation logic from the requirement phase to the microservices phase, the methodology section proposed in the next section expands this entire concept to a comprehensive end-to-end pipeline, which includes system architecture, cloud, simulation, and performance evaluation.

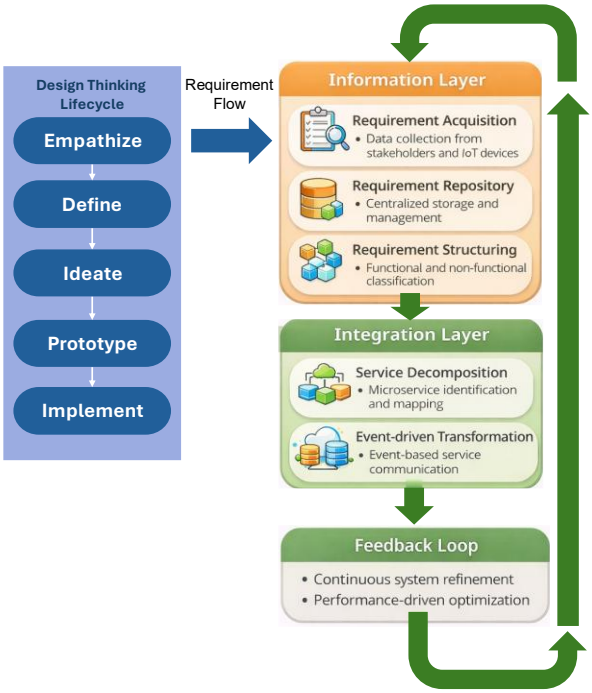

Figure 1. Design thinking-driven conceptual framework illustrating the transformation of healthcare requirements into fine-grained, event-driven microservices through structured information and integration layers, supported by an iterative feedback loop.

3. Research methodology

3.1. Overall methodology

The research methodology for the proposed research follows a structured and multi-layered approach that incorporates design thinking methodologies with microservices-based system engineering and cloud computing strategies. The proposed research methodology framework is designed to transform healthcare requirements into cloud-native services using a well-defined set of stages. As depicted in Figure 2, the proposed research methodology follows a hierarchical pipeline that starts with requirement acquisition and ends with the selection of the optimal deployment strategy. The framework is designed to specifically distinguish data flow, control flow, and feedback loops to ensure a linear approach with iterative enhancement. The proposed research methodology starts with requirement engineering and data acquisition, where requirements are gathered from various stakeholders such as patients, doctors, IoT devices, and healthcare databases. The requirements are then processed and transmitted using a well-defined pipeline to enable transformation into deployable services. The next step in the proposed research methodology framework is the incorporation of a domain-driven approach for system requirements analysis and decomposition into independent business domains. The domains are further divided into fine-grained microservices using the Single Responsibility Principle (SRP). The next step in the proposed research methodology framework is microservice architecture design. This system utilizes a hybrid model of communication that incorporates the use of REST synchronous and AMQP asynchronous messaging for efficient coordination between services. Moving on, the system is developed and deployed within a cloud environment, utilizing different paradigms such as monolithic SOA, containerized microservices on EC2, and serverless microservices on AWS Lambda. This is done for a comprehensive comparative evaluation of the system. Moving ahead, the methodology incorporates an experimental simulation of workload, where the

system is analyzed for its behavior based on stochastic conditions through Poisson distributions. This is done for a thorough evaluation of the scalability of the system, which is critical for a healthcare domain. Further, the framework incorporates a two-tier evaluation methodology, which is as follows: (1) Quantitative evaluation of the system's performance. (2) Expert opinion-based qualitative evaluation. Moving ahead, the methodology incorporates a comparative evaluation and selection of the best deployment strategy, where trade-offs between different factors such as performance, cost, scalability, and complexity are considered. It is imperative to state that this methodology is not linear, and this is depicted in Figure 1, where a feedback loop is also incorporated, which enables the results of the performance evaluation process to be used for refining earlier stages such as the design and deployment process.

3.2. Holistic System workflow and domain decomposition

In order to extend the conceptual foundation developed in the previous section to a system-level representation, a holistic workflow of the proposed framework is developed. This phase formalizes the end-to-end system operation flow of the framework. As shown in Figure 3, the framework follows a structured pipeline from stakeholder-driven demand acquisition to microservice development. The process commences with stakeholders, who include patients, clinicians, and healthcare providers. These stakeholders define the initial system demands. The demands include system operation requirements and domain constraint requirements in the healthcare domain. The acquired demands are then passed on to requirement engineers. The requirement engineers transform raw system demands into structured system requirements. At this phase of the framework, system requirements are structured into functional requirements (FRs) and non-functional requirements (NFRs). This ensures that system behavior and quality attributes are adequately represented. This structured representation of system requirements is equivalent to the Information Layer developed in Section 3.1. Once system requirements have been structured, the framework introduces a domain analysis phase. This phase involves mapping system requirements to business domains. This is done by performing a contextual service breakdown. This phase breaks down system requirements into coarse-grained functional domains. Each domain represents an important system function in the healthcare domain. The domains include patient management, consultation, notification, and sensor data processing. Once system requirements have been broken down into coarse-grained domains, they are then broken down into fine-grained microservices. Each microservice is designed to perform a single function. This process of decomposing the system into smaller parts is achieved with the assistance of domain experts and system architects to ensure that the services are well demarcated, have high cohesion, and are loosely coupled. To ensure the practicality of the framework, the execution environment has been divided into two domains: Local Domain, which refers to the front ends of the interfaces, including the browsers and the user interaction layers; and the Cloud Domain, which refers to the components of the system, including the middle layer, the service layer, the backend, and the storage of the data. The separation of the domains allows for the efficient distribution of the tasks of the system, taking advantage of the flexibility of the cloud computing environment. The framework has also adopted the use of lightweight protocols to enable the different components of the system to communicate with one another, as is the case in the modern microservices architecture approach. The protocols adopted are as follows: (1) The framework has adopted the use of REST and HTTP protocols to enable the different components of the system to communicate with one another in a synchronous manner. (2) The framework has adopted the use of the AMQP protocol, which allows the different components of the system to communicate with one another

in an asynchronous manner, enabling the different components of the system to communicate with one another in an event-driven manner. The data exchange protocols adopted by the framework are JSON, which allows for lightweight data exchange, and SOAP, which allows for the integration of legacy systems with the modern systems, if need be. The data of the operation of the system, including the patient information, the consultations, and the notification, are managed by the distributed database systems. From the perspective of the operation of the systems, the framework has adopted the DevOps approach to enable the rapid development, deployment, and scaling of the systems. Once the microservices are developed, they are deployed to the cloud, where they are integrated into the operation of the systems as independent components of the systems, and the computation, processing, and data management are achieved within the cloud environment, while the user interface is achieved through the conventional web interface.

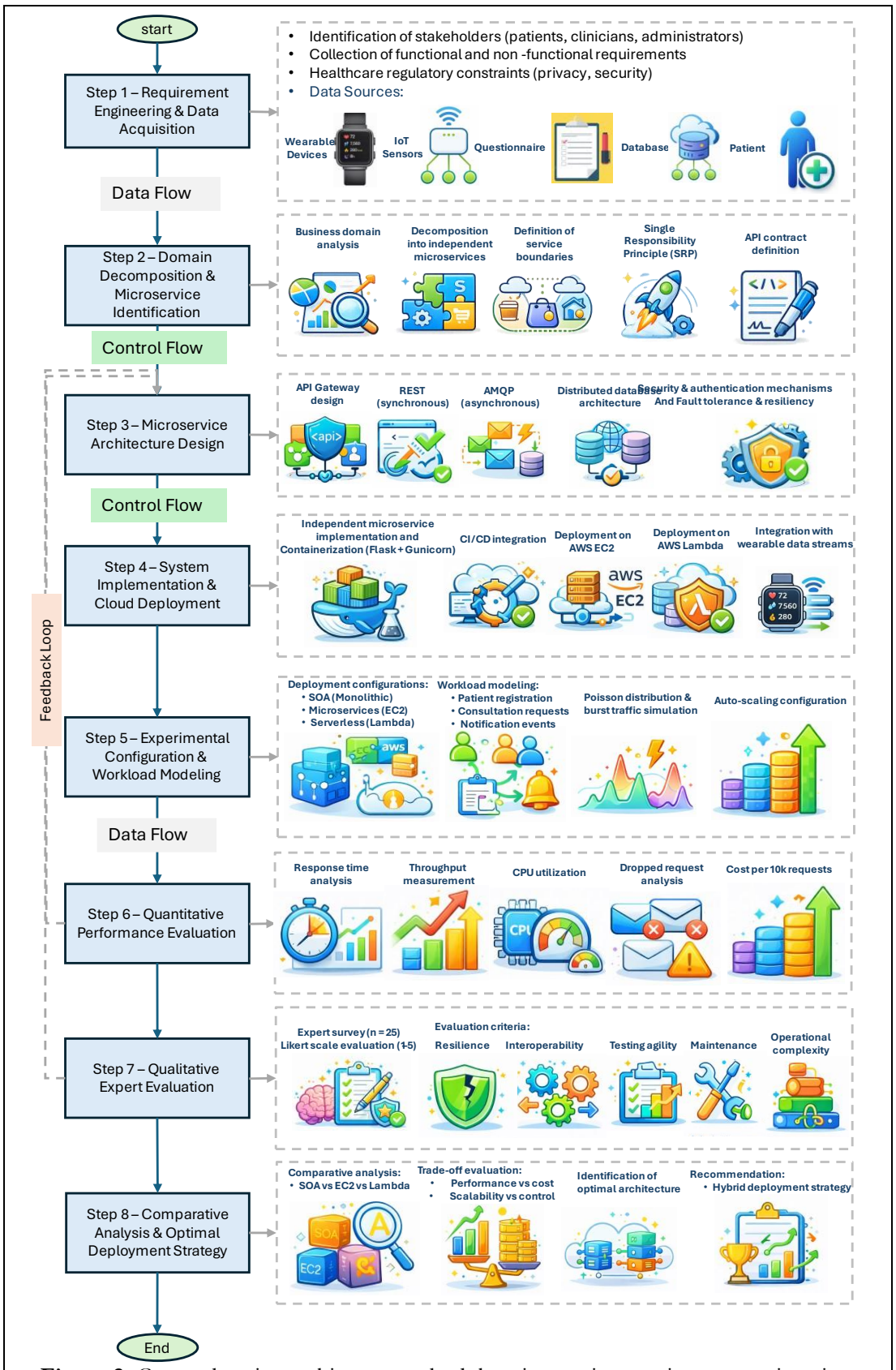

Figure 2. Comprehensive multi-stage methodology integrating requirement engineering, domain decomposition, microservice architecture design, cloud deployment, simulation, and evaluation with explicit data flow, control flow, and feedback loop mechanisms.

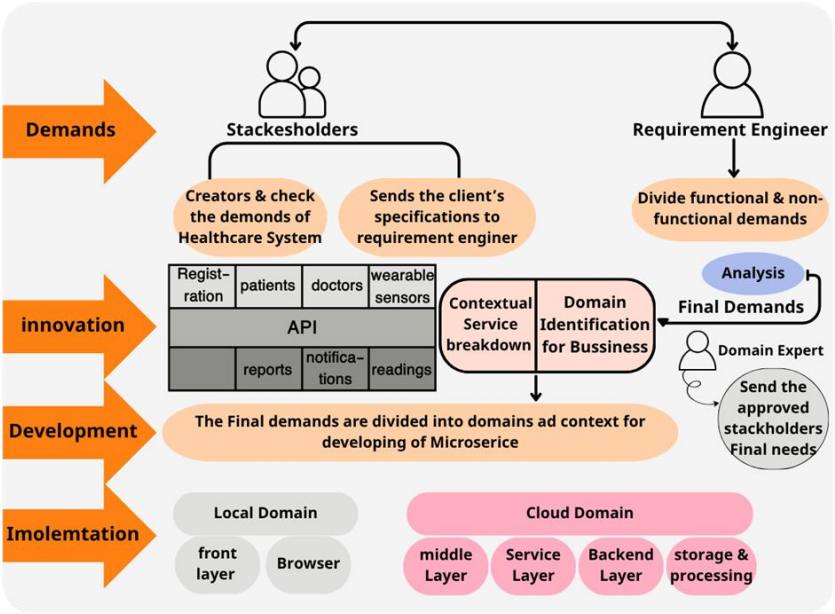

Figure 3. Holistic framework view showing stakeholder-driven requirement processing, domain identification, and decomposition into microservices across local and cloud environments

3.3. Microservice Architecture Design

This section is based on the design of the functional architecture of the proposed healthcare system, based on the conceptual framework and the holistic workflow developed in the previous sections. As discussed in the previous section, this section formalizes how the services are structured, interrelated, and deployed within a layering scheme. As depicted in Figure 4, the proposed healthcare system is based on a multi-tier microservice-based system that aims to ensure scalability, flexibility, interoperability, and fault isolation in a cloud-based healthcare system. As depicted in the figure, the proposed healthcare system is based on a four-tier microservice-based system. (1) The Service Consumer Tier: This tier represents the entry point of the system, where end-users and external systems will interact with the system. This tier will include web-based interfaces for patients, doctors, and hospital management systems, as well as IoT-based wearable devices. (2) The Service Access Tier: This is the gateway between the service consumer and the microservice, responsible for handling the communication between them through APIs. This tier will support different types of communication protocols depending on the different system requirements. For synchronous systems, the proposed healthcare system will be based on the following protocols: REST/HTTP for synchronous communication. For asynchronous systems, the proposed healthcare system will be based on the following protocol: AMQP for asynchronous communication. Additionally, the proposed healthcare system will support different data formats and protocols, including: (A) JSON and SOAP for structured data exchange. (B) MQTT, CoAP, and HL7 for IoT and healthcare systems, respectively [33]. This tier is responsible for handling user requests and sending them to the services through a standard interface, ensuring the accessibility and usability of the system for different user groups. (3) The Service Messaging Tier is responsible for facilitating communication among microservices using asynchronous messaging styles. As illustrated in Figure 4, this tier is implemented using Advanced Message Queuing Protocol (AMQP) for reliable service interaction. In this tier, the system is further decomposed into several domain-specific microservices, including: (A) Patient Registration Service. (B) Doctor Registration Service. (C) Consultation Services. (D) Notification Services. (E) Sensor Data Processing Services. Each of these services is

autonomous but interacts using message queues. This facilitates: (A) Loose Coupling. (B) High scalability. (C) Fault Isolation. (D) This ensures that service failure is not cascaded across the system. (4) The Data Management Tier is responsible for providing storage and data management services for the system. As expected, each microservice maintains an independent database, making it suitable for microservices-based systems. This is also in line with microservices architecture, which supports distributed data management. This tier facilitates: (A) Storage of patient records and medical history. (B) Management of consultation data and notification services. (C) Handling of sensor data. The use of distributed database systems enhances system scalability.

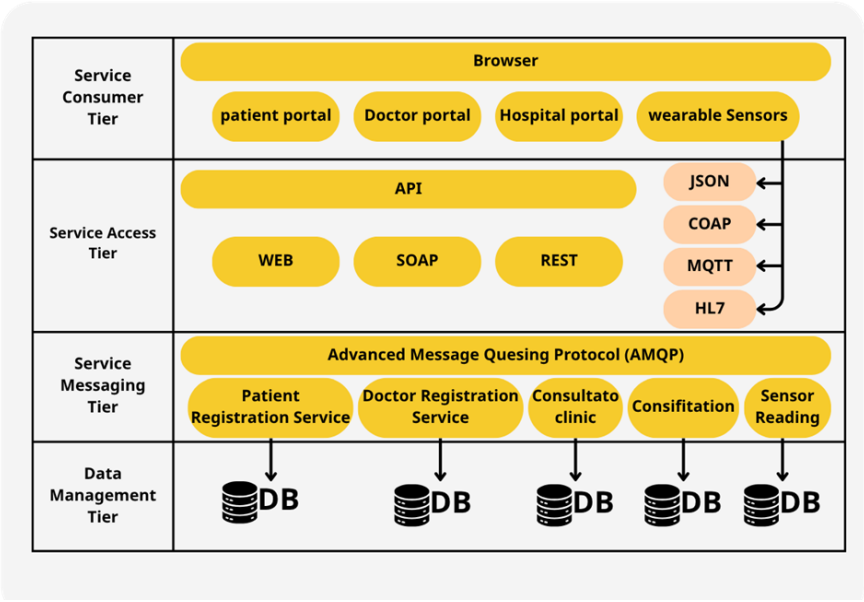

Figure 4. Functional view of the healthcare microservice architecture showing service consumer, service access, messaging, and data management tiers

3.4. Deployment Architectures

In order to verify the feasibility of the proposed framework under different cloud computing environments, the system is deployed using three different architectural models. This is expected to provide a more reliable platform for evaluating the proposed framework under different system design approaches. The use of three different system deployment models is expected to allow for a more comprehensive comparative evaluation of different system design approaches. The three different system deployment models adopted for this research are: (A) Service-Oriented Architecture-based monolithic system deployment on EC2. (B) Microservices-based system deployment on EC2. (C) Serverless microservices-based system deployment using AWS Lambda.

A. SOA-Based Monolithic Deployment

In this case, the system is implemented as one cohesive application where all the different functional components, such as the handling of patients, the provision of consultative services, and the handling of notifications, are tightly coupled. The application is deployed on a single Amazon EC2 instance. Although this deployment strategy makes the deployment process simpler and reduces the overall complexity of the deployment process, it also has significant limitations in terms of flexibility, scalability, and failure isolation.

B. Microservices Deployment on EC2

In the second deployment paradigm, the overall healthcare system is decomposed into different microservices, where each of the different functions of the healthcare system is implemented as a separate microservice, and the different microservices are deployed as separate EC2

instances. This deployment strategy makes the overall healthcare system more flexible, scalable, and maintainable, as the different microservices can be updated separately according to the demands of the workload.

C. Serverless Microservices using AWS Lambda

In the third deployment strategy, the overall healthcare system is implemented as serverless microservices, where the different functions of the healthcare system are implemented as separate AWS Lambda functions, which are event-driven, meaning that the overall healthcare system can scale automatically according to the number of incoming events .In this deployment strategy, the overall healthcare system is more scalable and cost-efficient, as the overall cost of deployment is directly proportional to the number of events, and the event-driven nature of the AWS Lambda functions makes the overall healthcare system more efficient, as the AWS Lambda functions can scale automatically according to the number of events.

D. Experimental Setup and Deployment Strategy

To assess the performance of the proposed framework, a detailed experimental setup has been designed. In this context, the framework is deployed on the basis of three different architectural styles: (1) SOA-based Monolithic Deployment on EC2, (2) Microservices Deployment on EC2, and (3) Serverless Microservices Deployment on AWS Lambda. This experimental setup facilitates a comparative analysis of different deployment styles with similar workload conditions. Monolithic style offers simplicity in terms of deployment, whereas the system lacks in terms of scalability. In contrast, the Microservices-based EC2 deployment offers greater flexibility and better scalability, though with increased operational complexity.

3.5. Workload Modeling and Simulation

To simulate the realistic behavior of a healthcare system, a detailed workload modeling approach has been adopted in the proposed framework, where stochastic process modeling is used for modeling the workload patterns in the healthcare system. In this context, the proposed framework includes: (1) Patient registration, (2) Consultation requests, and (3) Notification triggers, which are modeled using Poisson distribution for simulating bursty traffic patterns in the healthcare system. In addition, auto-scaling features are also included in the proposed framework.

3.6. Performance Evaluation and Validation

To evaluate the performance of the proposed framework, a detailed performance evaluation approach has been adopted, where the proposed framework is evaluated on the basis of quantitative as well as qualitative performance evaluation criteria. In this context, the proposed framework is quantitatively evaluated on the basis of: (1) Quantitative Evaluation: (A) Response Time, (B) Throughput, (C) CPU Utilization, (D) Dropped Request Rate, and (E) Cost for 10,000 requests, whereas the proposed framework is qualitatively evaluated on the basis of a detailed expert survey, where 25 experts are surveyed with a Likert scale (1-5) for evaluating: (1) Quantitative Evaluation: (A) Resilience, (B) Interoperability, (C) Testing Agility, (D) Maintenance, and (E) Complexity.

3.7. Comparative Analysis and Optimal Strategy Selection

Lastly, a comparative analysis is conducted to select the best approach for the deployment process. The analysis includes: (1) performance-cost, and (2) scalability-control. Consequently, an optimal hybrid approach for the proposed system, considering the advantages of EC2-based microservices and serverless computing, is proposed.

4. Case Study: SaaS-Based Healthcare Application

As proposed in the system, a SaaS-based healthcare application layer is included with the aim of facilitating continuous patient monitoring and intelligent clinical decision-making through the integration of wearable sensing technologies and cloud-based analytics. As depicted in Figure 5, the proposed system facilitates real-time acquisition, processing, and utilization of patient health information within a unified digital

healthcare ecosystem. In the proposed system, at the application layer, patient information acquisition occurs through wearable health sensors and user interfaces, which include patient mobile applications and portals, followed by the transmission of information to the Personal Health Record (PHR) system for storage and analysis. The information acquisition process includes a variety of physiological and behavioral parameters, which include vital signs, activity, sleep, glucose, ECG, medication, and environment, among others. This process enables comprehensive patient profiling, thereby facilitating proactive healthcare interventions. As depicted in Figure 5, the proposed system enables patient interaction with the system through a Decision Support System (DSS) in which adaptive medical queries and questionnaires are used for decision-making purposes. These answers are accumulated and processed along with sensor information to provide relevant insights. This information is further provided to healthcare practitioners to enable decision-making. The basic functionality of the proposed system is implemented through a series of independent microservices. Each microservice is designed to perform a single responsibility. The proposed system's basic functionality is implemented through the following microservices: (1) User Service: This service handles user registrations and their profiles, including patients, physicians, and specialists. (2) Consultation Service: This handles the entire consultation process between the patient and the healthcare practitioner. (3) Patient Care Service: This enables the continuous remote monitoring and clinical evaluation of the patient's condition. This can be particularly useful for providing personalized medical care for the patient after surgery. (4) Notification Service: This enables the communication between the system components by notifying the patient upon the submission of medical information and the physician for providing feedback and recommendations. Through this proposed application framework, the challenges faced by modern healthcare systems can be effectively addressed. This is done by providing real-time monitoring and decision-making capabilities through the proposed system's functionality. This will improve the overall efficiency and effectiveness of the healthcare system.

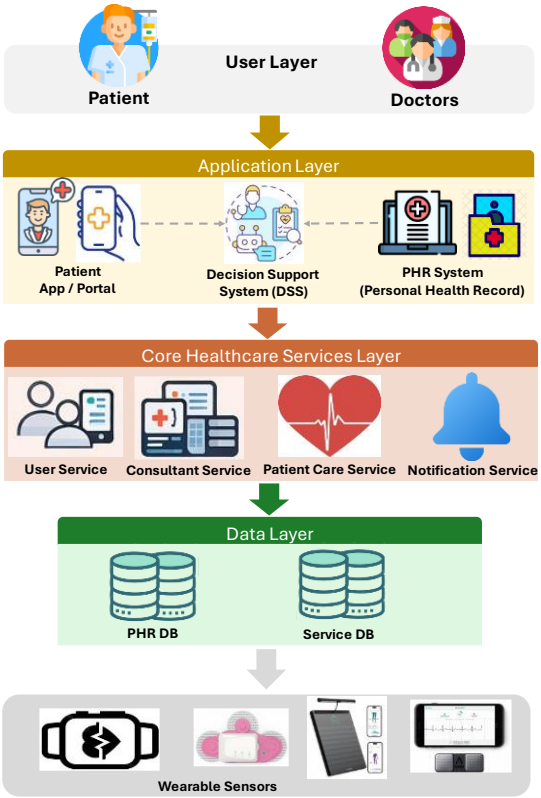

Figure 5. Layered SaaS-based healthcare application architecture illustrating user interaction, application processing, microservice orchestration, data management, and IoT-based wearable sensing integration

5. Results

In this section, the designed framework of the healthcare system using the fashionable microservice-based framework will be independently evaluated using empirical methods. A comparison of the three paradigms of deployments will be made with a realistic healthcare workload, as described in the experimental evaluation: (1) Monolithic deployments as a Service-Oriented Architecture (SOA). (2) Microservices running on long-term Amazon EC2 hosts. (3) Micro-services as serverless Lambda functions on AWS. Containerized Microservices (Flask + Gunicorn), a locally simulated Amazon RDS-like stored, and AWS Lambda Events 512 MB Memory Allocation clouds were given to Linux-based EC2 T3. medium hardware (2 vCPU, 4 GB RAM) for use as a simulated environment. A basic threshold-based controller was used for simulating EC2 autoscaling policies (up to 6 instances). Lambda concurrency limits were set for a comparable level with standard mid-tier accounts. An HTTP request was sent from a simulated client, which sent a request after a Poisson process for the sustained load and burst case. The metrics used for evaluation were average response time, request throughput, CPU utilization, requests that were dropped, and cost for 10,000 requests in the cloud.

5.1. Average Response Time

The latency characteristics of all three models are summarized in Figure 6. As can be seen, there is a clear performance gradient among all models. The serverless Lambda model achieved the minimum average response time of 0.98 s, followed by the EC2-based microservices model, which reported an average latency of 1.05 s. On the other hand, the SOA monolithic model reported the highest average latency of 1.45 s. Although it might seem like a small difference, it is significant in healthcare systems, especially for

user-facing services, which are often characterized by short, repetitive, and latency-sensitive interactions. The poor response time of the SOA monolithic model can be explained from different perspectives. As can be understood from Figure 4, the SOA monolithic model consists of different functional modules, all of which share the execution environment. This means that when there is a traffic spike for a specific service, it can indirectly affect other parts of the system, especially because of increased contentions for shared CPU, memory, and I/O resources, especially when there is a mix of both reads and writes. This means that, unlike other models, the monolithic model is not capable of isolating latency-sensitive service execution from other, more resource-intensive services. The larger execution footprint of the monolithic model also means increased coordination costs, which can explain the poor response time. On the other hand, the EC2 microservices model reported better performance because of service decomposition and isolation. Each service can execute requests more independently because of the architectural design, which means that there is little or no interference from unrelated services. This explains why the EC2 microservices model was able to reduce average latency by about 27.6%, i.e., (1.45 - 1.05)/1.45. Although EC2 reported better performance, it is clear from Figure 6 that it is still slower than the serverless Lambda model. This is because EC2 is constrained by resource allocation, service orchestration, network latency, and horizontal scaling. The serverless Lambda model reported better latency because of the combination of fine-grained event execution and horizontal scaling. As can be observed from the evaluated workload, the request types are mostly short-lived and independently executable, which is consistent with the execution semantics for serverless computing. This means Lambda avoids the inefficiencies of keeping instances running for variable workloads. This makes Lambda a good fit for healthcare application event patterns such as notifications, sensor-driven updates, and asynchronous service calls. The result shown in Figure 6 therefore validates the argument for a potentially better latency profile for serverless-based microservices for applications where the service invocations are mostly bursty and fine-grained rather than long-lived and stateful.

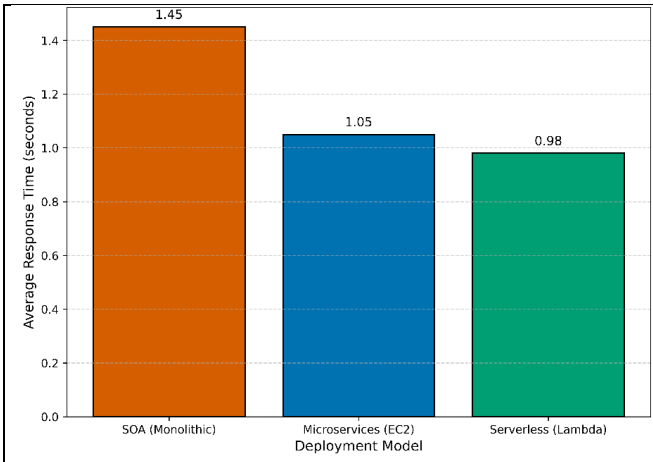

Figure 6. Average response time across deployment architecture

5.2. Throughput and Scalability

Figure 7 illustrates the achieved throughput for the increased levels of concurrent requests. As demonstrated, there are three scalability envelopes, with the SOA monolith achieving a maximum of 900 requests/s, the EC2 microservices model achieving 1400 requests/s, and the Lambda-based microservices architecture achieving the highest rate of 2400 requests/s. This is consistent with the architectural scalability expectations developed in the methodology section, where the principles of decomposition,

independent scalability, and event-driven execution were considered critical components. The SOA monolith's maximum achievable rate also suggests that this type of architecture is prone to early resource saturation. This is due to the monolithic execution, where all components compete within a shared runtime environment. As the level of concurrency increases, any bottlenecks experienced within a particular component will eventually propagate across the entire application boundary. This implies that the SOA monolith is not efficient in handling increased levels of concurrent healthcare transactions, particularly where components such as registration, consultation, and notification are active within a shared process space. The EC2 microservices model has a higher rate due to the principle of decomposition, where the increased load is not concentrated within the application's core components. Instead, each component is allowed to consume resources independently, thereby providing a higher rate of scalability and lower architectural inflexibility. Compared with SOA, the EC2 architecture has a higher peak throughput by about 55.6%, i.e., (1400 – 900)/900. This verifies the importance of fine-grained service partitioning and confirms that microservice decomposition is not just a design preference, but also a scalability enabler for systems operating in a variety of healthcare workload conditions. However, the best results are achieved with Lambda microservices, which support a peak rate of 2400 requests/s, i.e., an increase in throughput by about 166.7% compared with SOA and 71.4% compared with EC2 microservices. The linear scalability of Lambda microservices is also evident from the growth pattern shown in Figure 7. This suggests that Lambda has a wider scalability range for all the concurrency levels used in this experiment. This is also evident from an architectural point of view, where serverless microservices scale not only faster but also differently compared with virtual machine-based microservices. Instead of scaling up by adding new instances, serverless microservices scale up by replicating functions in reaction to events. This is particularly useful for healthcare systems, where the workload is highly variable, but the unit of work is also very fine-grained. This is not only shown by the higher peak rate, but also by the scalability pattern shown in Figure 7, which does not indicate that Lambda is "faster" than the others, but rather that serverless microservices are a different scalability model, which is also a much better fit for highly variable and event-driven healthcare systems. This is why Lambda is best used for transactional front-end services, asynchronous notifications, and any other parts of the application that need to scale up dynamically.

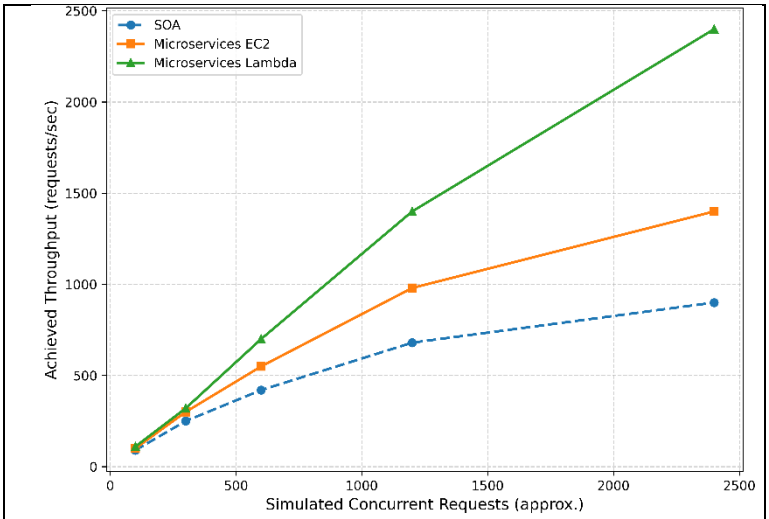

Figure 7. Measured throughput (requests per second) as the simulated concurrent load increases.

5.3. Aggregated Quantitative Metrics

The overall quantitative comparison is consolidated in Table 2, which serves as the main numerical reference for cross-model interpretation. A careful analysis of Table 2 reveals that a very important structural pattern is hidden in the data: it is impossible to determine the best architecture based on a particular metric; instead, the entire set of deployment paradigm performance-cost-operability trade-off profiles should be considered. The SOA model always underperforms in throughput and dropped requests, even though it benefits from lower average CPU utilization than EC2. This is a very important observation: lower CPU utilization for SOA should not be considered a positive attribute; instead, it should be considered a symptom that the architecture fails to translate resource utilization into performance throughput under high concurrency. In other words, the problem is not that the SOA architecture utilizes high resources; instead, it fails to efficiently convert that high utilization into performance throughput under high concurrency. The EC2 microservices architecture benefits from the highest average CPU utilization (68%), which may imply that instance-based microservices have a higher baseline operation overhead. This may be explained by the nature of always-on microservices running inside containers that continuously consume resources even if the workload changes over time. Still, this high CPU utilization is accompanied by better throughput and lower dropped requests than SOA; therefore, the EC2 microservices architecture may be considered a better trade-off between performance and cost. The Lambda microservices architecture benefits from the best response time, highest throughput, lowest dropped requests, and lowest estimated cost for executing 10k requests. This is a particularly important observation: often, in cloud computing environments, better performance is accompanied by a higher cost; however, the Lambda microservices architecture benefits from both high performance and lower cost due to the elasticity and pay-per-use billing principles that are particularly well-suited for a high variability workload pattern. The numerical data presented in Table 2 supports the statement that serverless microservices have the best trade-off for the healthcare workload under consideration.

Table 2. Summary of quantitative metrics across deployment models.

Metric	SOA (Monolith)	Microservices EC2	Microservices Lambda	Unit	Notes
Avg. Response Time	1.45	1.05	0.98	s	Average over mixed workload
Throughput (peak)	900	1400	2400	req/s	Peak sustainable throughput observed
CPU Utilization (avg)	35	68	55	%	Average across application nodes
Dropped Requests	1600	320	150	count	Total dropped during stress tests
Est. Cost /10k reqs	1.20	0.85	0.65	USD	Estimated operational cost proxy

5.4. CPU Utilization and Resource Behavior Under Burst Load

Figure 8 presents the temporal CPU utilization of the three architectures. The figure presents a more dynamic view of the results shown in Table 2. Figure 8 presents the CPU utilization of each architecture over time as the load intensity changes. The EC2 microservices architecture continues to show the highest CPU utilization range. The CPU utilization of EC2 microservices

increases over time and eventually peaks at approximately 66% by the end of the simulation. This suggests that EC2 microservices handle the load by continuing to increase the CPU utilization of the provisioned resources. Although this is not a bad practice per se, it suggests that handling bursts of load within EC2 microservices requires continued autoscaling of the instance types used. The SOA monolithic architecture shows unstable CPU utilization. The CPU utilization of SOA increases over time as it handles the load. However, it still performs worse and experiences higher request loss. This suggests that unstable CPU utilization does not necessarily map well to improved request handling. The CPU utilization of SOA is unstable because it bundles all the services together. The CPU utilization of one service affects the performance of other services. In healthcare workloads, this is particularly undesirable, as critical operations such as notification of the patient or updating the consultation should not be slowed down by congestion elsewhere in the system. The Lambda line continues to be lower and more horizontal, remaining in the low thirties percentage range. This suggests that there is a completely different resource model at work here, rather than providing the high utilization on the long-running hosts, the Lambda architecture distributes the execution across the short-running managed invocations. The reduced CPU usage profile is not necessarily indicative of weakness, but rather the elasticity of the architecture and the abstraction of the managed infrastructure. Figure 8, in effect, suggests that the Lambda architecture is able to handle the bursty nature of the workload without subjecting the application to the same level of infrastructure usage as the EC2 and SOA scenarios.

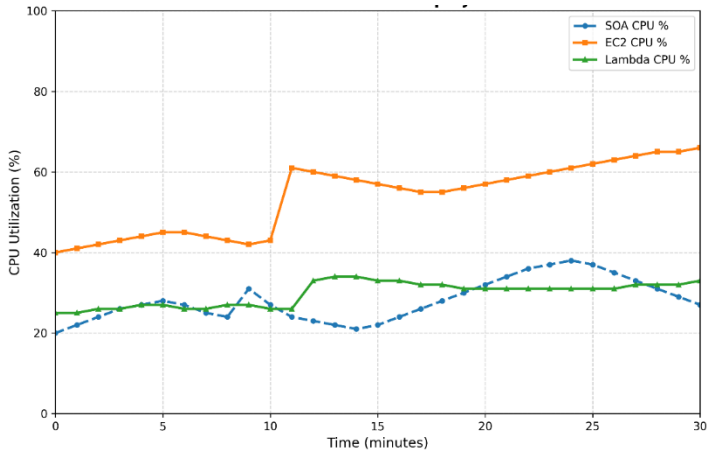

Figure 8. Observed CPU utilization across deployment models during a simulated traffic burst.

5.5. Reliability and Request Loss Analysis

Dropped requests are one of the most critical result variables of this study due to their direct relation to system reliability and potential impact on patient safety and efficiency in a healthcare system. The behavior of dropped requests is depicted in Figure 9 (Number of dropped requests as load increases; serverless shows lower drop rates at high concurrency). As depicted in the figure, the SOA architecture exhibits the steepest drop rate curve. This means that SOA architecture exhibits approximately 7200 dropped requests when concurrency reaches its peak. This result indicates that the monolithic model fails to maintain transaction integrity once concurrency surpasses its sustainable capacity. This escalation is nonlinear, and reliability declines drastically after a certain threshold. This is a critical weakness in a system designed for a healthcare system, where overload must be handled with graceful degradation rather than catastrophic failure. The EC2 microservice architecture significantly improves reliability by reducing the number of dropped requests to approximately 1800 when concurrency peaks. This improvement verifies

our assumption that microservice architecture and independent runtime contexts reduce request drop due to their limited scope of impact. However, this result also indicates that request drop is still an issue with instance-based scaling. This result is critical due to the lag between increased concurrency and increased capacity. This lag is observable during sudden increases in concurrency. The Lambda architecture exhibits the best reliability results of all architectures tested. This result is due to fewer dropped requests during peak concurrency. Specifically, this result indicates approximately 500 dropped requests during peak concurrency. This means that serverless outperforms SOA by over 93% and EC2 by approximately 72%. This result indicates a fundamental difference in handling system overload. Figure 9 therefore indicates that serverless deployment is significantly better than both SOA and EC2 architectures in terms of reliability.

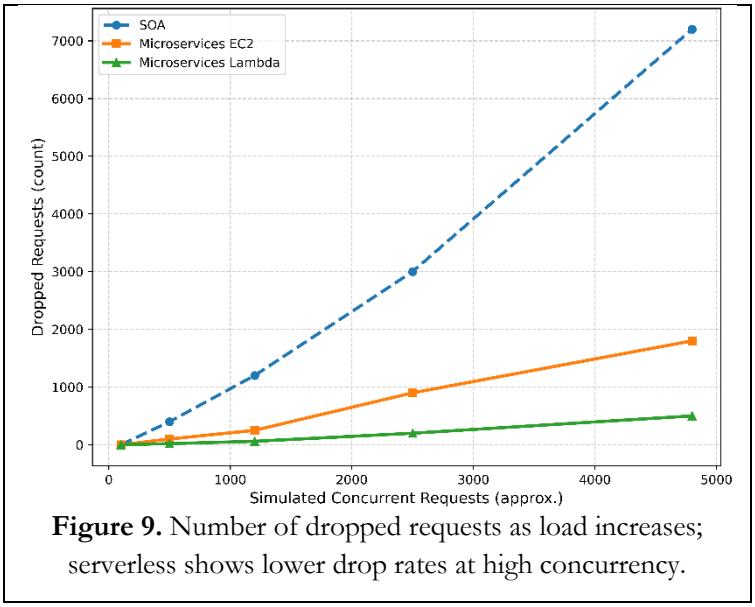

Figure 9. Number of dropped requests as load increases; serverless shows lower drop rates at high concurrency.

5.6. Cost Analysis

As can be seen in Figure 10, which presents the comparative cost structure, the key difference in this section from the previous ones, which emphasized performance, is that here we are concerned with the cost distribution between different components in each architecture. As can be expected, in the SOA and EC2 architectures, the cost is primarily related to the compute layer, as these architectures are based on provisioned and running environments. This can be particularly well observed in the EC2-based architecture, where performance is improved due to service isolation, which, however, results in a number of running service instances, containerization, and underlying infrastructure management. Thus, in the case of EC2-based microservices, performance is improved, but the organization still pays for underlying persistence, regardless of whether demand is constant or not. In the Lambda architecture, the cost related to the compute layer is significantly reduced, with a corresponding increase in the cost related to other components, namely, storage and networking. This corresponds well with the nature of serverless environments, where the cost is not related to underlying idle instances, but rather to the invocation-based execution and underlying cloud services. This approach is particularly well suited for bursty healthcare requests, which are not constant over time. The result in Figure 10 above supports the cost analysis in Table 2, where Lambda architecture results in the lowest cost, i.e., 0.65 USD for 10k requests, as opposed to 0.85 USD for EC2-based microservices and 1.20 USD for SOA-based architecture. Significantly, this result should not be construed as implying that Lambda is generally cheaper than EC2 in all circumstances. Rather, Lambda is cheaper with respect to a given workload profile, which features variability, event-driven processing, and

spikes. In a more constant, high-duty-cycle workload, the cost effectiveness of Lambda relative to EC2 might be different. This adds greater depth to the discussion by considering cost effectiveness in terms of the workload, not just the architecture itself.

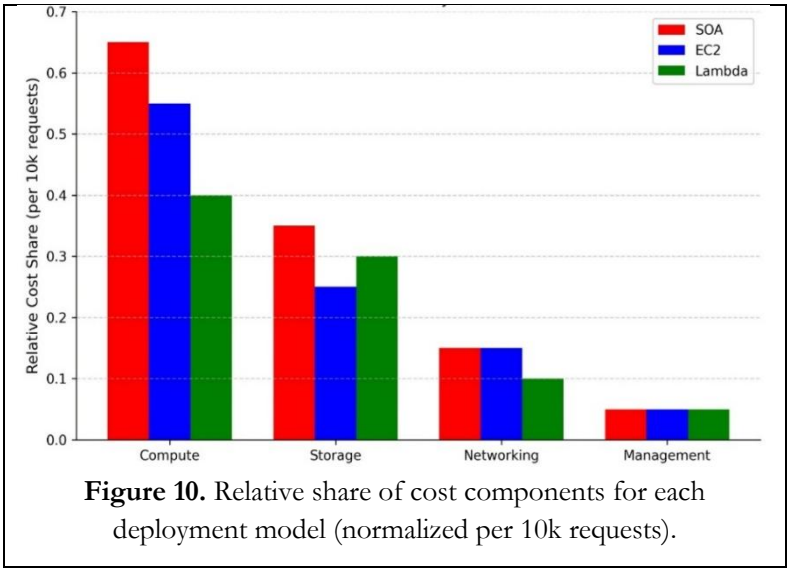

Figure 10. Relative share of cost components for each deployment model (normalized per 10k requests).

5.7. Qualitative Expert Evaluation

As a complement to the quantitative study, a qualitative evaluation approach was also implemented through a structured expert survey. A total of 25 experts from various domains, including requirements engineers, software architects, testers, and operations engineers, were consulted for the survey. The survey aimed to evaluate the six success factors using a 1-5 Likert scale. The success factors were Requirements Management, Interoperability, Resilience, Testing Agility, Operational Complexity, and Maintenance. The expert evaluation means are shown in Figure 11. The expert evaluation means are listed numerically in Table 3: Aggregated expert evaluation scores across key architectural success factors. The expert evaluation means show a high consistency rate compared to the quantitative evaluation. The SOA model had the lowest expert evaluation means for all six success factors. This is consistent with the quantitative evaluation results for the SOA model. In addition, the SOA model had the lowest evaluation means for the Resilience and Testing Agility success factors, indicating the perceived lack of agility and resilience for the SOA model. In contrast, the two microservices-based approaches had much higher evaluation means than the other two approaches. This is consistent with the quantitative evaluation results. This indicates the perceived benefits of the two microservices-based approaches for the EC2 and Lambda environments. The Lambda approach had slightly higher evaluation means than the EC2 approach for nearly all success factors. In particular, the Lambda approach had higher evaluation means than the EC2 approach for the Resilience and Maintenance success factors by 0.4 and 0.3 units, respectively. This is a very interesting result from a conceptual point of view. This result suggests that the benefits of managed infrastructure are not limited to performance benefits but can also improve the perceived maintainability of the system. This is an important point for discussion. While the evaluation means for the two approaches were different for nearly all success factors, the difference between the two approaches was not extreme for some success factors. In particular, the evaluation means for the Interoperability success factor were only 1 unit apart between the two approaches. This is an interesting point for discussion. This point for discussion avoids the evaluation results from being overly simplistic. While the infrastructure can impact the system's operability, other qualities may also be heavily influenced by the implementation discipline and interface design.

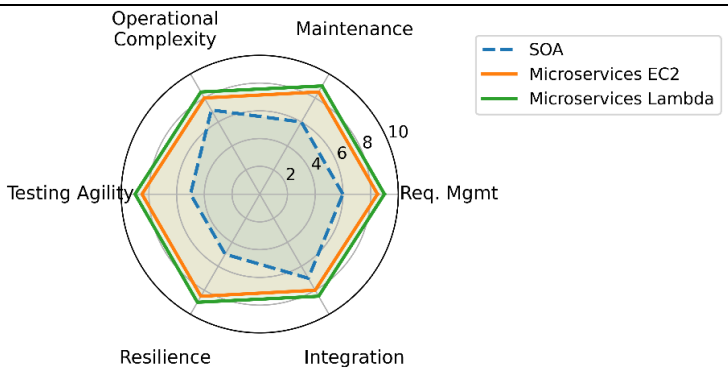

Figure 11. Radar-style visualization of experts aggregated mean scores for six success factors.

Table 3. Aggregated expert means scores (1-5 Likert).

Success Factor	SOA	Microservices (EC2)	Microservices (Lambda)
Requirements Management	3.0	4.1	4.3
Interoperability	3.2	3.8	3.9
Resilience	2.4	4.0	4.4
Testing Agility	2.5	4.0	4.3
Operational Complexity	3.1	3.5	3.8
Maintenance	2.9	4.2	4.5

5.8. Summary of expert opinions (Q&A)

In addition to the quantitative evaluation, a qualitative evaluation of the system was performed. A total of 25 experts contributed to the evaluation of this system. The participants of this evaluation were software architects, DevOps, system analysts, and healthcare IT. The main goal of this evaluation was to obtain experiential opinions about the system, especially regarding the practical aspects of different deployment models. A semi-structured questionnaire in the form of Q&A was employed for this purpose. This type of evaluation helps in understanding the system better, especially regarding aspects such as maintainability, system resilience, and operational complexity, which are difficult to measure quantitatively. The opinions of the experts were grouped into six major aspects of system architecture, including requirements traceability, interoperability, system resilience, testing agility, operational complexity, and maintainability. The summarized opinions of the experts are provided in Table 4.

From the qualitative data presented in Table 4, we may conclude that there is a constant correlation with the quantitative data; however, additional insights are provided for the reasons behind the performance differences. Based on all the dimensions considered, experts showed a distinct preference for the microservices paradigm over the traditional SOA approach, especially for areas related to system flexibility, robustness, and maintainability. Concerning the requirements traceability (Q1), experts stressed that the microservices paradigm provides a significant enhancement for the system's modularization capabilities, where complex healthcare processes are segmented with distinct boundaries for service components. It is also mentioned that a serverless approach should have better governance in order to avoid a fragmented architecture that may lead to a loss of architectural integrity. With respect to interoperability (Q2), experts' responses indicated that the actual compatibility level is less related to the architectural paradigm adopted and more related to API design best practices. Nevertheless, the microservices paradigm promotes the development of fine-granular versioned APIs that

may improve interoperability compared to monolithic SOA architectures that often have tightly coupled APIs. Finally, for the dimension related to system robustness (Q3), we may conclude that this is one of the most critical areas where experts showed a strong preference for the EC2 and Lambda microservices architectures due to their ability to isolate failures that may cause system breakdowns. On the contrary, the SOA approach is considered to be extremely weak in this area, where a failure in one component may cause a complete system breakdown. Lambda was marginally favored due to its built-in retry features and fault tolerance. In the case of testing agility (Q4), microservices were once again favored as they allow unit and contract testing. The experts pointed out the advantage of faster development and more frequent testing. However, it was noted that Lambda added complexity to the integration testing of microservices, especially while handling distributed event-driven systems. The complexity of operational aspects (Q5) shows a critical trade-off between EC2 and Lambda. Although Lambda greatly simplifies infrastructure management, it was noted by the experts that this simplicity might actually hide complexities. These complexities include debugging and handling cold starts. On the other hand, EC2 requires more explicit DevOps activities, but it was noted by the experts that it also offered more transparency and control over the system. Finally, regarding the maintainability of the systems (Q6), a hybrid approach was recommended by the experts. EC2 microservices were seen as more suitable for stable and high-throughput workloads, such as analytics and continuous processing. On the other hand, Lambda was seen as more suitable for event-driven and bursty workloads, such as notifications and alerts. This further confirms the result that no deployment model is universally best and that a choice of deployment models should be workload-dependent. Overall, the qualitative results not only confirm the quantitative results but also offer practical guidelines for designing systems and show the importance of balancing scalability, operational aspects, and complexity while designing healthcare cloud applications.

Table 4. Summary of expert qualitative insights across architectural paradigms.

Question ID	Evaluation Dimension	Expert Insight Summary
Q1.	Requirements Traceability	Experts agree that using microservices greatly improves requirements tracking, because it separates domain contexts into separate, clear services. As for Lambda, it reduces the operational load, but it needs a strong governance system so that there is no distraction or chaos in the services
Q2.	Interoperability	Experts said that there is no big difference in compatibility from a technical standpoint, because it depends more on the design of the API itself, but microservices encourage the use of small APIs divided into versions (versioned APIs), and this facilitates integration compared to a single SOA system.
Q3.	Resilience	Opinions were in favor of Lambda and EC2 microservices in terms of flexibility. Lambda received positive reviews because it provides automatic fault isolation and auto-retry, while EC2 offers predictable and easy-to-track failure areas. As for the single SOA system, it received a lower rating because any failure in it could cause a cascading collapse of the entire system.
Q4.	Testing Agility	Flexibility in testing has improved dramatically with microservices, because unit and contract testing has become simpler and more frequent. It is true that Lambda adds some complexity to integration

		testing, but it simplifies deployment testing because each function is tested independently.
Q5.	Operational Complexity between EC2 and Lambda	EC2 needs more follow-up from the DevOps team, such as capacity planning and container orchestration, while Lambda greatly reduces infrastructure management. But experts warn that Lambda's apparent simplicity could mask greater complexity in tracking distributed errors or dealing with cold starts.
Q6.	Long-term maintainability	Experts advised using a hybrid strategy, where EC2 microservices are used in parts that have a constant and large load, such as analytics, and Lambda in tasks that are incidental or rarely used, such as emergency notifications or temporary events (notification bursts).

5.9. Tail Latency and Autoscaling Efficiency

Apart from average latency, the distribution of latency is also evaluated using the cumulative distribution function, as depicted in Figure 12. This figure is of critical interest because average response time might not reveal issues when there is a problem of tail latency, especially in systems where response times are sometimes extremely slow. The SOA curve is shifted to the right, and it also indicates a larger tail, which means more requests are experiencing increased latency. This indicates that even if average response time is within acceptable limits, the system is more vulnerable. In healthcare systems, tail events are more critical than average degradation because they are usually associated with outliers like urgent notification or completion of tasks. Both EC2 microservices and Lambda microservices have better CDFs, which indicates a sharp increase in their curves. This indicates better request handling, and it is also evident that there is a lesser chance of extreme latency. The Lambda microservices have a better CDF, which again indicates that serverless microservices are better for healthcare systems because of their better response time, as also indicated in Figure 6, which showed that serverless microservices are better for healthcare systems because, apart from better average response time, latency stability is also better.

5.10. Autoscaling Efficiency

Finally, the last dimension of performance is the efficiency of autoscaling in response to increased throughput requirements. This is shown in Figure 13. This figure shows a key difference between the two scaling approaches: the increased delay between the scaling request and the scaling response. This delay is more pronounced for the EC2 approach compared to the Lambda approach. In the case of the EC2 approach, the delay is more than 75 seconds for high throughput requirements. This delay is a result of the various steps involved in scaling instances, including the launch of containers and the registration of services. In the case of the Lambda approach, the delay is much less than 5 seconds for high throughput requirements. This shows that the Lambda approach is more efficient in handling sudden changes in throughput requirements. The quantitative results from Figures 8 and 9 also show this. This is a key advantage for the Lambda approach over the EC2 approach. The disadvantage for the Lambda approach is the cold start problem for the invocations. However, from Figure 13, it can be seen that the serverless approach is more efficient compared to the instance-based approach for the healthcare application.

If the results of the quantitative and qualitative analysis are considered together, a clear trend is evident. The SOA monolithic architecture is worst performing in all but one of the critical metrics. This validates the conclusion that the monolithic model is not appropriate for healthcare applications that exhibit mixed workloads, event-driven models, and burst-driven service delivery patterns. The EC2 microservices

architecture is a clear second best, providing superior scalability, modularity, request loss, and expert-rated maintainability. However, this is offset by increased CPU utilization and a greater need for infrastructure management. Therefore, the EC2 microservices model is a strong contender when the organization has a need for predictable long-running runtime environments, greater control over the underlying infrastructure, or a need for stateful services. The Lambda serverless microservices architecture provides the best overall profile for any of the microservices models considered in this work. It has the best average response times, highest throughputs, lowest request loss, lowest cost estimate, and highest qualitative ratings. This suggests that serverless deployment is highly beneficial for healthcare systems that are driven by burst-driven, short-lived, and event-driven service delivery models. However, as discussed in the qualitative and autoscaling sections, this does not preclude the possibility that a serverless model may not be appropriate for all healthcare applications, particularly if they exhibit stateful sessions, state coordination, or specialized long-running service delivery. This provides a clear architectural recommendation that serverless microservices are the best option for burst-driven and event-driven healthcare services, whereas the EC2-based microservices model is appropriate for stateful, constantly active, or infrastructure-sensitive service components. This provides a clear rationale for the hybrid deployment model that is discussed later in this paper.

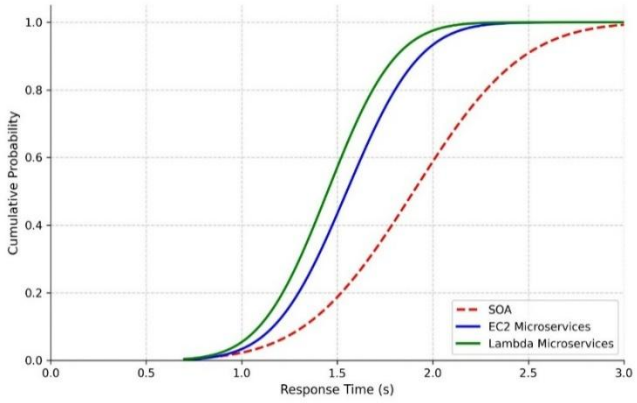

Figure 12. Cumulative distribution function (CDF) of response times showing tail behavior differences.

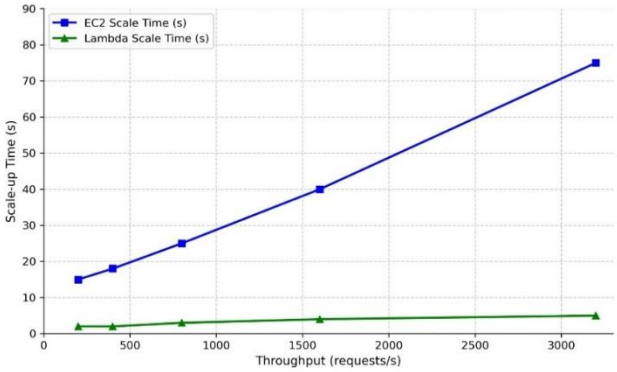

Figure 13. Time required for the deployment model to scale in response to increased throughput demand.

6. Summery and conclusion

This paper has presented a comprehensive evaluation of three prominent deployment architectures—SOA monolithic architecture, EC2-based microservices architecture, and serverless Lambda microservices architecture—in

the context of a smart healthcare system. By employing both quantitative performance evaluation and qualitative expert analysis, this research aims to provide a holistic understanding of the implications of these architectures on the efficiency, scalability, and reliability of a smart healthcare system. From the experimental evaluation, it is evident that traditional SOA architectures are inefficient for handling the high-concurrency requirements of a smart healthcare system. The SOA architecture recorded the highest latency (1.45s), lowest throughput (900 req/s), and highest dropped requests (1600). These findings validate the inefficiency of traditional SOA architectures in handling distributed workloads that are characteristic of modern healthcare systems. Microservices architectures have been found to have a significant positive effect on the performance of a smart healthcare system. EC2-based microservices architectures have been found to provide stable performance with a fair trade-off between throughput (1400 req/s) and latency (1.05s). However, this comes at the cost of increased CPU utilization (68%) and increased operation costs due to infrastructure management overheads. Among the three architectures considered in this research, serverless microservices architecture (Lambda) has been found to be the most efficient in the context of a smart healthcare system. Lambda recorded the lowest latency (0.98s), highest throughput (2400 req/s), and lowest dropped requests (150). Moreover, Lambda has been found to provide the most cost-effective solution (0.65 USD for every 10k req/s). This is due to the event-driven execution paradigm employed by Lambda and its ability to scale horizontally rapidly.

Despite the comprehensiveness of the evaluation approach, certain limitations must be considered with respect to the results of this study. Firstly, the experimental approach used in this study assumes a simulated healthcare environment that attempts to emulate real-world scenarios with respect to patient registration, consultation requests, and notification services. Although this approach attempts to emulate real-world scenarios with respect to steady-state and burst traffic patterns, in a real-world healthcare environment, certain other factors, such as heterogeneous user behavior, network instabilities, and other external constraints, could significantly affect the performance of the system. Secondly, with respect to the serverless architecture, this study assumes a standard AWS Lambda concurrency as defined for a mid-tier cloud account. In a real-world environment, these could be significantly different due to certain other factors, such as account configuration, service availability, and service level agreements, which could affect the scalability benefits of the Lambda-based approach. Thirdly, with respect to network latency, the experimental approach used in this study assumes a controlled environment with respect to network latency, which could be a significant factor in a real-world environment, particularly for a microservices-based approach, where communication between different components of the system could be significant, particularly with respect to service discovery mechanisms and API gateway-based communication overheads. With respect to cost, the results of this study are based on assumed pricing models, which are based on standard pricing models for cloud-based service providers. Although this approach could be a reasonable assumption with respect to a typical cloud-based environment, certain other factors, such as region-specific pricing, reserved instance pricing, and other long-term pricing agreements, could be significant in a real-world environment. As such, the cost efficiency of each deployment model should be viewed as suggestive rather than definitive. The limitations outlined above suggest that, while the results provide strong evidence of the relative performance of different paradigms, further validation of the results in a "real-world" healthcare context is necessary.

Following the results obtained through this research, further research directions include improving the flexibility, realism, and security of microservices-based healthcare systems. In this context, one of the research directions includes the incorporation of predictive autoscaling techniques, such as machine learning-based models. These models have the capability to

predict workload changes and thus reduce cold-start latency of serverless platforms. In addition, further research directions include the evaluation of hybrid microservices orchestration models, including the dynamic distribution of workloads between EC2-based microservices and AWS Lambda-based serverless functions. These models are expected to achieve the best possible performance, cost, and control by making deployment decisions in a highly adaptive manner. Furthermore, one of the important research directions includes the evaluation of the proposed framework using real-world healthcare datasets, such as electronic health records (EHRs). The incorporation of real-world patient information is expected to provide a better evaluation of the proposed framework and ensure the applicability of the proposed microservices-based healthcare architecture. In addition, further research directions include a higher focus on the evaluation of security and privacy-aware deployment models. These include the incorporation of protection mechanisms at the boundaries of each microservice and compliance with healthcare regulations such as HIPAA and GDPR. These are important factors to ensure the protection of patient information within a microservices-based healthcare environment. Overall, this research is expected to transition the proposed framework from a controlled environment to a highly adaptive and secure microservices-based healthcare platform.

References

J. Ruokolainen et al., "Digital healthcare platform ecosystem design: A case study of an ecosystem for Parkinson's disease patients," Technovation, vol. 120, Feb. 2023, doi: 10.1016/j.technovation.2022.102551.

J. Zaki, S. M. R. Islam, N. S. Alghamdi, M. Abdullah-Al-Wadud, and K. S. Kwak, "Introducing Cloud-Assisted Micro-Service-Based Software Development Framework for Healthcare Systems," IEEE Access, vol. 10, pp. 33332–33348, 2022, doi: 10.1109/ACCESS.2022.3161455.

R. T. Avireneni, "Cloud-Native Middleware: Architecting Scalable and Resilient Healthcare Delivery Systems100 1. The Evolution of Healthcare Infrastructure 1.1 From Legacy Systems to Cloud-Native Paradigms," 2025, doi: 10.32996/jcsts.

F. M. Garcia-Moreno, M. Bermudez-Edo, J. L. Garrido, E. Rodríguez-García, J. M. Pérez-Mármol, and M. J. Rodríguez-Fórtiz, "A microservices e-health system for ecological frailty assessment using wearables," Sensors (Switzerland), vol. 20, no. 12, pp. 1–23, Jun. 2020, doi: 10.3390/s20123427.

M. Ianculescu and A. Alexandru, "Microservices-A Catalyzer for better managing healthcare data empowerment," Studies in Informatics and Control, vol. 29, no. 2, pp. 231–242, 2020, doi: 10.24846/V29I2Y202008.

A. D. Walle et al., "Intention to use wearable health devices and its predictors among diabetes mellitus patients in Amhara region referral hospitals, Ethiopia: Using modified UTAUT-2 model," Inform. Med. Unlocked, vol. 36, Jan. 2023, doi: 10.1016/j.imu.2022.101157.

H. Calderon-Gomez et al., "Telemonitoring System for Infectious Disease Prediction in Elderly People Based on a Novel Microservice Architecture," IEEE Access, vol. 8, pp. 118340–118354, 2020, doi: 10.1109/ACCESS.2020.3005638.

L. Nguyen Tran Thanh et al., "IoHT-MBA: An Internet of Healthcare Things (IoHT) Platform based on Microservice and Brokerless Architecture." [Online]. Available: https://oauth.net/2/

F. Subhan et al., "AI-Enabled Wearable Medical Internet of Things in Healthcare System: A Survey," Applied Sciences (Switzerland), vol. 13, no. 3, Feb. 2023, doi: 10.3390/app13031394.

I. Shabani, T. Biba, and B. Çiço, "Design of a Cattle-Health-Monitoring System Using Microservices and IoT Devices," Computers, vol. 11, no. 5, May 2022, doi: 10.3390/computers11050079.

A. Garcés-Jiménez et al., "Medical prognosis of infectious diseases in nursing homes by applying machine learning on clinical data collected in cloud microservices," Int. J. Environ. Res. Public Health, vol. 18, no. 24, Dec. 2021, doi: 10.3390/ijerph182413278.

M. Golec et al., "BlockFaaS: Blockchain-enabled Serverless Computing Framework for AI-driven IoT Healthcare Applications," J. Grid Comput., vol. 21, no. 4, Dec. 2023, doi: 10.1007/s10723-023-09691-w.

J. Sadek, D. Craig, and M. Trenell, “Design and Implementation of Medical Searching System Based on Microservices and Serverless Architectures,” in Procedia Computer Science, Elsevier B.V., 2021, pp. 615–622. doi: 10.1016/j.procs.2021.12.056.

A. Jangra and N. Mangla, “An efficient load balancing framework for deploying resource schedulingin cloud based communication in healthcare,” Measurement: Sensors, vol. 25, Feb. 2023, doi: 10.1016/j.measen.2022.100584.

I. K. Aksakalli, T. Celik, A. B. Can, and B. Tekinerdogan, “Systematic Approach for Generation of Feasible Deployment Alternatives for Microservices,” IEEE Access, vol. 9, pp. 29505–29529, 2021, doi: 10.1109/ACCESS.2021.3057582.

H. Calderón-Gómez et al., “Evaluating service-oriented and microservice architecture patterns to deploy ehealth applications in cloud computing environment,” Applied Sciences (Switzerland), vol. 11, no. 10, May 2021, doi: 10.3390/app11104350.

J. Islam, T. Kumar, I. Kovacevic, and E. Harjula, “Resource-Aware Dynamic Service Deployment for Local IoT Edge Computing: Healthcare Use Case,” IEEE Access, vol. 9, pp. 115868–115884, 2021, doi: 10.1109/ACCESS.2021.3102867.

D. Loconte et al., “Serverless Microservice Architecture for Cloud-Edge Intelligence in Sensor Networks,” IEEE Sens. J., vol. 25, no. 5, pp. 7875–7885, 2025, doi: 10.1109/JSEN.2024.3502254.

R. K. Deka, A. Ghosh, S. Nanda, R. K. Barik, and M. J. Saikia, “Smart Healthcare System in Server-Less Environment: Concepts, Architecture, Challenges, Future Directions,” Apr. 01, 2024, Multidisciplinary Digital Publishing Institute (MDPI). doi: 10.3390/computers13040105.

A. N. Gohar, S. A. Abdelmawgoud, and M. S. Farhan, “A Patient-Centric Healthcare Framework Reference Architecture for Better Semantic Interoperability Based on Blockchain, Cloud, and IoT,” IEEE Access, vol. 10, pp. 92137–92157, 2022, doi: 10.1109/ACCESS.2022.3202902.

A. Gamal, N. E. Elsayed, and A. Saber, “Interoperable electronic medical record Framework based on Microservice architecture,” Online, 2025. [Online]. Available: https://ejai.journals.ekb.eg/1

“Design and Evaluation of Secure Microservices Architecture for HIPAA-Compliant Prescription Processing on AWS and OpenShift,” International Journal of Artificial Intelligence, Data Science, and Machine Learning, vol. 5, no. 2, Jun. 2024, doi: 10.63282/3050-9262.ijaidsml-v5i2p116.

“Cloud Microservices for Secure Health Data Access,” International Journal of Emerging Trends in Computer Science and Information Technology, vol. 6, no. 4, Oct. 2025, doi: 10.63282/3050-9246.ijetcsit-v6i4p104.

K. Zala, H. K. Thakkar, R. Jadeja, P. Singh, K. Kotecha, and M. Shukla, “PRMS: Design and Development of Patients’ E-Healthcare Records Management System for Privacy Preservation in Third Party Cloud Platforms,” IEEE Access, vol. 10, pp. 85777–85791, 2022, doi: 10.1109/ACCESS.2022.3198094.

M. Harshith et al., “Federated microservices architecture with blockchain for privacy-preserving and scalable healthcare analytics,” Sci. Rep., vol. 16, no. 1, p. 9023, Feb. 2026, doi: 10.1038/s41598-026-39837-1.

M. G. do Nascimento, J. M. N. David, M. A. R. Dantas, R. Braga, and V. Ströele, “An e-health environment conceived with the support of a self-adaptive IoT architecture,” Internet of Things (The Netherlands), vol. 36, Mar. 2026, doi: 10.1016/j.iot.2025.101844.

M. Maray, S. M. Rizwan, E. Mustafa, and J. Shuja, “Microservices enabled bidirectional fault-tolerance scheme for healthcare internet of things,” Cluster Comput., vol. 27, no. 4, pp. 4621–4633, Jul. 2024, doi: 10.1007/s10586-023-04192-7.

B. Havano and M. Morozov, “ASSESSING THE HUMAN CONDITION IN MEDICAL CYBER-PHYSICAL SYSTEM BASED ON MICROSERVICE ARCHITECTURE,” ADVANCES IN CYBER-PHYSICAL SYSTEMS, vol. 6, no. 2, p. 2021, 2021, doi: 10.23939/acps2021.__.___.

N. Kodakandla, “MICRO SERVICE-BASED MIDDLEWARE ARCHITECTURE FOR REAL-TIME DATA PROCESSING IN IOT ENVIRONMENT,” 2023. [Online]. Available: www.ijnrd.org

M. Zamani and S. Sharifian, “Distributed edge to cloud ensemble deep learning architecture to diagnose Covid-19 from lung image in IoT based e-Health system,” Journal of Supercomputing, vol. 80, no. 13, pp. 18492–18520, Sep. 2024, doi: 10.1007/s11227-024-06163-0.

R. Ouyang et al., “A Microservice and Serverless Architecture for Secure IoT System,” Sensors, vol. 23, no. 10, May 2023, doi: 10.3390/s23104868.

J. M. García-Campos, A. M. Alcaide, A. Letrado-Castellanos, R. Portillo, and J. I. Leon, “A Scalable Microservices Architecture for Condition Monitoring and State-of-Health Tracking in Power Conversion Systems,” Sensors, vol. 26, no. 4, Feb. 2026, doi: 10.3390/s26041282.

R. M. Masoud, R. M. A. Bakir, M. S. Saraya, and S. M. Ayyad, “BREAST-CAD: A Computer-Aided Diagnosis System for Breast Cancer Detection Using Machine Learning,” Technologies (Basel)., vol. 13, no. 7, p. 268, 2025.