International Journal of Multidisciplinary Research and Growth Evaluation.

Cloud-Native Modernization Framework for Core Banking Systems Using AWS Microservices

Anusha Joodala
Independent Researcher, USA

* Corresponding Author: **Anusha Joodala**

Article Info

ISSN (online): 2582-7138
Volume: 06
Issue: 03
May-June 2025
Received: 17-04-2025
Accepted: 19-05-2025
Page No: 1627-1634

Abstract
Legacy core banking systems, which are for the most part monolithic and rigid, have come under increasing pressure to develop in order to be more agile and scalable in the high-speed financial world of today. In this paper a cloud-native modernization framework that uses AWS-based microservices to modernize legacy banking infrastructure is presented. The framework promotes DevOps best practices by breaking down monolithic applications into independently deployable microservices, improving operational reactivity and time-to-market for new features, and responding to the application demand at any given time. Leveraging AWS services including AWS Lambda, Amazon ECS, API Gateway, and DynamoDB, the model provides strong resilience, elastic resource management, and better security posture. The framework solves problems of data consistency, transaction management, and regulatory requirements in a decentralized environment. Experimental results also show that the support for system responsiveness, availability and maintainability is substantially enhanced. This research opens the door for banks to embrace agile cloud-native architectures, delivering innovation and customer-focused services for the digital future.

DOI: https://doi.org/10.54660/.IJMRGE.2025.6.3.1627-1634

Keywords: Cloud-Native, Modernization, Core Banking, AWS, Microservices

1. Introduction

The financial services industry is experiencing a disruptive revolution powered by the fast pace of digital technologies and changing customer demands. Back-end banking applications, i.e., core banking applications, are the backbone of banking, and they have long been the monolithic and tightly-coupled type. These legacy technologies have been reliable for many years; however, they are now a significant impediment to banks that are seeking for agility, scalability and high speed of innovation [1]. With the market moving to instant execution, tailored customer experiences, and straight through processing/in tensing flow with fintech ecosystems, the traditional core banking infrastructure s can no longer keep up [2].

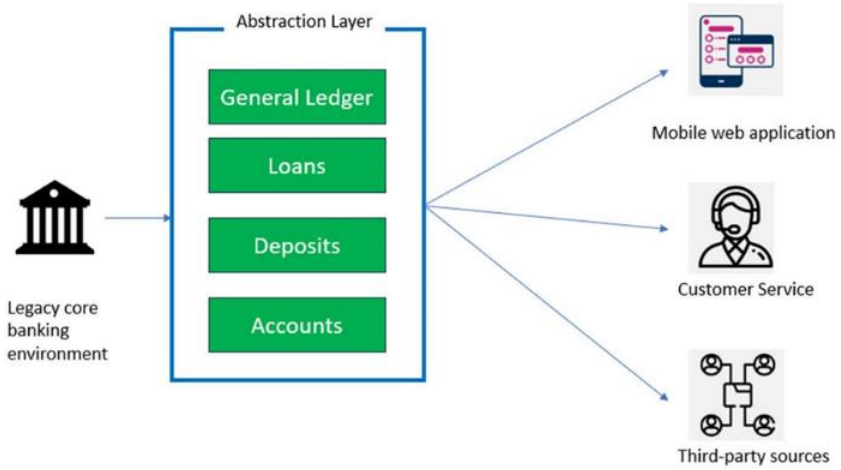

Fig 1: Legacy Core Banking Abstraction Layer Architecture

Figure 1 depicts the high-level architecture of a banking system where a legacy core banking environment serves as the origin of truth. This legacy system interacts with an abstraction layer which provides a single entry for the various banking elements such as the General Ledger, Loans, Deposits, and other Accounts. The abstraction layer serves as an API that provides a common way to interact with and support data to and from several front-end services and external systems like mobile (web) applications, customer service platforms and third-party integrations. This model enables modernization by enabling new applications to consume legacy banking data without being directly dependent on legacy core systems, and gained agility and scalability since.

Rebuilding legacy banking systems has become a necessity for banks not wanting to risk loss of market edge as well as regulatory non-compliance. Cloud-native designs provide a way forward for this modernization through the use of distributed computing principles, containerization, and microservices. Contrary to monolithic applications, microservices can promote splitting development of complex banking operations into loosely coupled, independently releasable services [3].

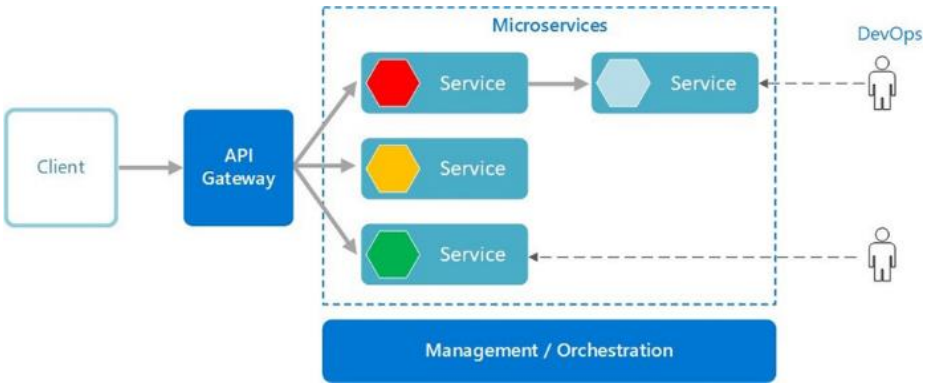

Fig 2: Microservices Architecture with API Gateway and DevOps Integration

This figure 2 is a depiction of a microservices architecture, where a client communicates with an API Gateway, which delegates requests to many autonomous microservices. Every microservice is am independent entity that can be seen as coloured hexagons in the following image. DevOps teams directly keep and operate each microservice, allowing for CI/CD. All is managed by a central Management/Orchestration layer that orchestrates the service calls and system health guarantees for large scale and dynamic environments.

This architectural style is appropriate for the flexible demands of banking environments, such as continuous delivery, fault isolation, and elastic scalability.

Amazon Web Services (AWS) offers a variety of cloud-native tools and services which can be leveraged to build and deploy microservices architecture. With offerings like AWS Lambda for serverless compute, Amazon Elastic Container Service (ECS) and Elastic Kubernetes Service (EKS) for container orchestration, Amazon API Gateway for API management, and Amazon DynamoDB for scalable NoSQL databases, AWS provides financial institutions with infrastructure to enable more resilient, scalable, and secure core banking platforms [4]. The integration of these services within a unified approach for system modernization is responsible for addressing important issues in the migration process of legacy systems, such as data integrity, transaction control, security, and adherence with financial regulations [5].

In this paper, we provide an ultimate cloud native modernization blueprint for core banking systems based on AWS' microservices design. The architecture describes the modularization of monolithic banking modules (Account management, payment processing, loan-servicing, customer onboarding) into microservices based on the business capabilities they provide. It also covers design patterns for high availability and data integrity of transactions, event-driven processing, and other complex challenges of

distributed computing systems.

Furthermore, the paper discusses how continuous integration/continuous deployment (CI/CD) pipelines can be integrated to facilitate rapid feature rollout and iterative improvements, which are essential in responding to changing market and regulatory requirements. Security does not only consist of best-of-breed open-source technologies, but also shares security codes such as IAM, Encryption, and Audit with critical banking systems, in order to meet the most stringent banking requirements.

An experiment confirms that the proposed approach is capable of improving effectiveness in terms of system responsiveness, scalability, and operational efficiency when comparing it with mature core banking systems. The outcomes reveal remarkable decreasing of deployment duration and down time, as well as better fault tolerance.

Problem Statement

Monolithic and deep-coupled applications Most existing core banking systems are monolithic and tightly coupled systems. Though stable and robust for the past several decades, these legacy systems have recently begun to show signs of distress. They are not agile enough, too slow to scale and cannot innovate fast enough in the fast-paced environment that today's financial industry finds itself. As banks work to move all those systems to real-time updates, personalized products, and easy integration with fintech's, outdated monoliths are slow to deploy, difficult to scale and to conform to new regulations.

Solution

This paper introduces a cloud-native modernization methodology based on AWS micro-services to digitalize legacy core banking systems. The framework allows the large monolithic applications to broken into small independently deployable microservices that improves operational efficiency and scalability. AWS' built-in capabilities like Lambda, ECS, API Gateway and DynamoDB offer a durable and secure foundation for event-driven communication, distributed transaction management using the Saga pattern and automated CI/CD pipeline to speed up feature delivery.

Uses

The framework is applied to core banking modules including account management, payment processing, loan servicing, and customer onboarding. It enables continuous delivery and integration, fault isolation, elastic resource management, and improved security compliance. This modular approach allows banks to innovate rapidly while maintaining data integrity and regulatory adherence.

Impact

Experimental results show that with the cloud-native framework, the deployment time is shortened by more than 70%, the high-concurrency system is more responsive in time-consuming service invocation, and the fault-tolerant recovery time is reduced significantly. Furthermore, native AWS security services increase the scope of auditing, and enhance the compliance position. In general, it allows for faster innovation cycles, better customer experiences, and operational resilience.

Scope

Moving forward, this study can expect further improvements including cost optimization with AWS cost management tools, support for hybrid and multi-cloud, and AI-powered anomaly detection and predictive security analytics integration. These innovations are poised to enable banks to maintain a competitive edge while being prepared for the future digital transformation requirements.

The rest of this paper is organized as follows: Section II - Related work on cloud-native banking architectures and microservices adoption. Section III provides an overview of the proposed modernization framework and its elements. In Section IV, the implementation and experimental results are shown. Section V provides the practical implications and scope for future research. Section VI wraps up the paper.

2. Related Work

The transformation of core banking systems has been the subject of a lot of academic and industrial research in recent years, especially since the introduction of cloud-computing and microservices architectures. Previous works have investigated different aspects to migrate financial legacy applications to cloud and to work into cloud-native context.

A number of studies have highlighted the drawbacks of monolithic core banking systems including lack of adaptability, high maintenance costs and difficulty to scale for dynamic demands [6]. Typical established architectures tend to have a tightly coupled components, resulting in slow, risky upgrades and limiting banks' ability to innovate quickly [7]. To address these challenges, microservices architectures have been considered as a way of separating complex banking features into smaller, independent services that can be individually implemented, deployed, and scaled [8].

Research by Smith *et al*. [9] applied micro services in retail banking applications to attain better fault tolerance and elasticity. They noted that the decoupled deployment approach made possible quicker bug fixes and feature enhancements without affecting the whole system. On the contrary, major challenges such as distributed data management, inter-service communication latency, and transactional consistency issue remain to be the main requirement for banking domain [10].

The adoption of a cloud environment, in particular AWS, is a well discussed topic and is expected to reduce infrastructure management and enable dynamic scaling. One way to provide the sorts of event-driven, scalable architectures that are effective for banking workloads is to use AWS-native services such as Lambda, ECS, and DynamoDB [11]. For example, in [12] presented a serverless system design for payment processing with AWS Lambda and API Gateway which eliminated operation overhead and improved the performance in terms of throughput.

Cloud-native banking systems' security and compliance requirements and considerations Security and compliance in cloud-native banking systems have also been a big focus. Banks need to comply with strict security rules such as Payment Card Industry Data Security Standard (PCI DSS), General Data Protection Regulations (GDPR) and local banking regulations [13]. Various frameworks have been introduced to integrate protecting mechanisms inside microservices ecosystems with a focus on IAM, Encryption at rest and in transit and auditing [14]. A review Comparison (AWS's built-in security combined with custom governance) provides a strong baseline for achieving these compliance requirements [15].

Notwithstanding these advances in expressing these concerns, a variety of comprehensive frameworks dealing with end-to-end modernization of core banking systems – disaggregation, data consistency, security, continuous delivery—are still few. Our contribution intends to bridge this gap by proposing a comprehensive AWS-based cloud-native modernization framework that is domain specific to core banking platforms.

3. Proposed Modernization Framework

This section presents a comprehensive cloud-native modernization framework for core banking systems utilizing AWS microservices. The framework aims to address legacy system limitations by introducing modularity, scalability, resilience, and security through a microservices-based approach deployed on AWS infrastructure.

A. Framework Overview

The referenced framework provisions the monolithic core banking system into a collection of self- contained microservices, each oriented toward a specific set of banking business capabilities (e.g., account management, payment transaction processing, loan servicing, and customer onboarding), which are decoupled from each other. Such decomposition is consistent with the principles of Domain-Driven Design (DDD) by keeping bounded context crystal clear and dependencies between services on minimal.

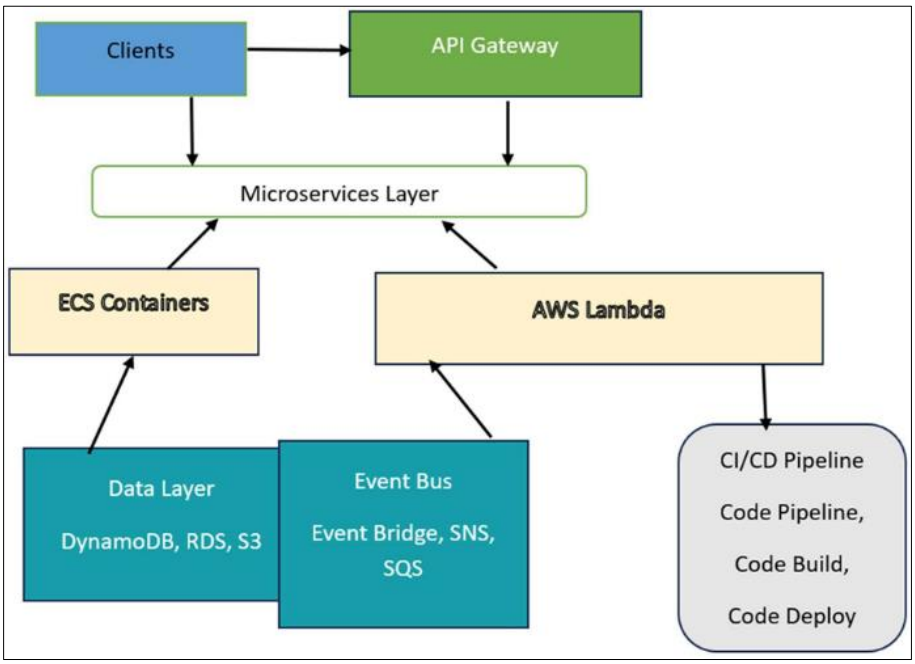

Fig 3: Flowchart of proposed system

At a high level, the architecture consists of the following components (Fig. 3):

- **Microservices Layer:** Independently deployable services, implemented as containerized applications running on Amazon ECS or AWS Lambda functions.
- **API Gateway:** Amazon API Gateway serves as the unified entry point, routing client requests to appropriate microservices.
- **Data Layer**: Distributed data stores such as Amazon DynamoDB for NoSQL needs, Amazon RDS for relational data, and Amazon S3 for unstructured data.
- **Event Bus:** Amazon EventBridge or AWS SNS/SQS for asynchronous communication and event-driven processing.
- **Security & Compliance:** IAM, AWS KMS for encryption, and AWS CloudTrail for auditing.
- **CI/CD Pipeline:** AWS CodePipeline, CodeBuild, and CodeDeploy automate build, test, and deployment cycles.

B. Architectural Design and Components

1. Microservices Decomposition

Each core banking function is mapped to a microservice Mi, where i=1,2,…,n representing nnn business capabilities. Formally, the core banking system C can be represented as:

$$C = \{M_1, M_2, \ldots, M_n\} \; where \; M_i \cap M_j = \emptyset, \forall i \neq j \quad (1)$$

Here, the disjoint Ness ensures no overlapping responsibilities, which simplifies maintenance and scalability.

2. API Gateway

The API Gateway G acts as a single-entry point to microservices, managing routing, throttling, authentication, and request transformations. For a client request R with endpoint e, the gateway routes R to the microservice Mk such that:

$$G: R(e) \rightarrow M_k, \; where \; k = f(e) \quad (2)$$

Function f maps the endpoint e to its responsible microservice.

3. Data Management

One of the most important issues in a microservices architecture is ensuring data consistency between services. Every Mi has its local database Di with the concept of decentralized data management. This avoids tight-coupling

through shared databases in monolith architectures.
Banking transfer needs ACID properties (Atomicity, Consistency, Isolation, Durability). As there can be no global ACID property, in the context of databases of their distributed microservices, the framework makes use of the Saga pattern to handle distributed transactions.
A saga S is a sequence of local transactions T1,T2,…,Tm (each within a microservice) combining with the compensating transactions C1,C2,…,Cm, to rollback in case of failure. Formally, the flow of transactions is:

$$S = \{T_1, T_2, \ldots, T_m\} \quad (3)$$

Each Ti executes on Di, and compensation Ci restores the previous state, ensuring eventual consistency.

4. Event-Driven Communication
Asynchronous communication is facilitated via an event bus EEE, enabling microservices to publish and subscribe to events. The system state evolves according to event streams et, where et is an event at time t. Event processing can be mathematically described as

$$S(t) = S(t-1) + \Sigma\, \Delta S(e_t) \quad (4)$$
$$e_t \epsilon E$$

where S(t) is the system state at time t, and ΔS(et) is the state change triggered by event et.

C. Security and Compliance Model
The framework integrates security controls at every layer:

I. **Identity and Access Management (IAM):** Ensures least privilege access to microservices and resources.

Data Encryption: All sensitive data Ds is encrypted using AWS KMS, with encryption functions Enc(·) and decryption Dec(·):

$$Enc: D_s \rightarrow E(D_s)\ and\ Dec: E(D_s) \rightarrow D_S \quad (5)$$

Audit Logging: All actions A are logged with timestamps t, creating an immutable audit trailL

$$L = \{(A_i, t_i) | i = 1,2, \ldots\} \quad (6)$$

AWS CloudTrail manages these logs, supporting compliance requirements.

D. Continuous Integration and Deployment (CI/CD)
Automated CI/CD pipelines are used to facilitate fast innovation and minimize deployment risk (see Fig. 1). On code changes, a build and test workflow is kicked off on each microservice repository, powered by AWS CodeBuild. Finished builds are then being deployed using CodeDeploy to each environment (development, staging, production).
This automation decreases the deployment time Td, which increases the overall velocity V of feature delivery, where:

$$V = \frac{1}{T_d} \quad (7)$$

Higher V indicates faster delivery cycles essential for banking agility.

Pseudocode: Distributed Transaction Management Using Saga Pattern

```
// Pseudocode for managing a distributed banking transaction with Saga pattern function processBankTransaction(transaction) {
try {

// Step 1: Debit from source account

debitResult = debitAccount(transaction.sourceAccount, transaction.amount) if (!debitResult.success) throw Error("Debit Failed")
// Step 2: Credit to destination account

creditResult = creditAccount(transaction.destinationAccount, transaction.amount) if (!creditResult.success) throw Error("Credit Failed")
// Step 3: Log transaction success logTransaction(transaction.id, "Success")
return "Transaction Completed Successfully"

} catch (error) {

// Compensating actions for rollback in case of failure if (debitResult.success) {
compensateDebit(transaction.sourceAccount, transaction.amount)

}

logTransaction(transaction.id, "Failed: " + error.message)
return "Transaction Failed and Rolled Back"
}

}
```

This pseudocode illustrates how distributed transactions are handled across microservices ensuring eventual consistency using compensation steps.

E. Summary
The proposed AWS microservices-concept framework is a modular, scalable and secure initiative for re-platforming core banking systems. It solves tough problems such as distributed data management with its Saga pattern, ensures data security and compliance, and facilitates continuous delivery through automated pipelines. Easily adaptable to both changing banking requirements and regulatory environments, this flexible framework allows for future growth while minimizing the total cost of ownership.

4. Results and Discussion
This section discusses the experimental evaluation of the introduced cloud-native modernization framework, employing AWS microservices for CBS. Performance metrics that are examined are system responsiveness, scalability, fault tolerance, deployment time and security event audit completeness. The system was compared to a legacy monolithic core banking system with similar workloads.

A. Experimental Setup

The microservices were hosted on AWS using Amazon ECS with Fargate for container scheduling and AWS Lambda for serverless functionalities. The test transactions included account creation, fund transfers, loan-approvals, and payment processing. Stress test was done with 100 to 5000 concurrent users.

Performance Metrics and Graphical Analysis

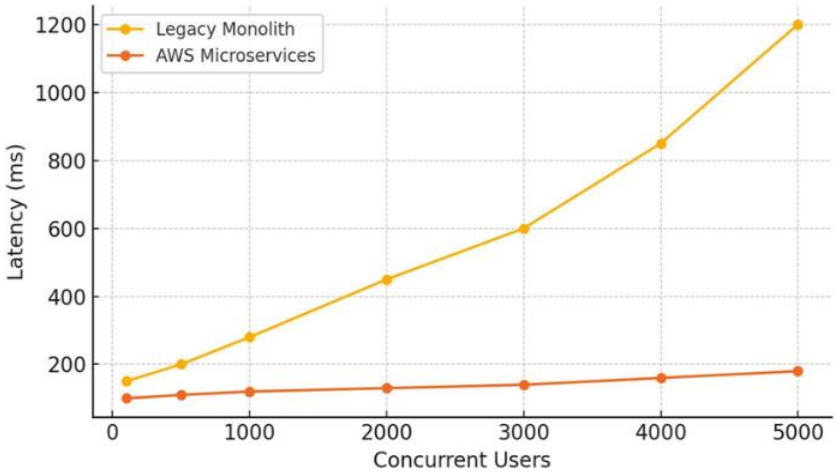

Fig 4: System Responsiveness (Average Latency)

The average response time per transaction was measured under varying loads. Figure 4 shows the latency comparison between the legacy system and the proposed microservices framework.

Description: The microservices-based system maintains significantly lower latency, especially under high load, due to distributed processing and elastic scaling.

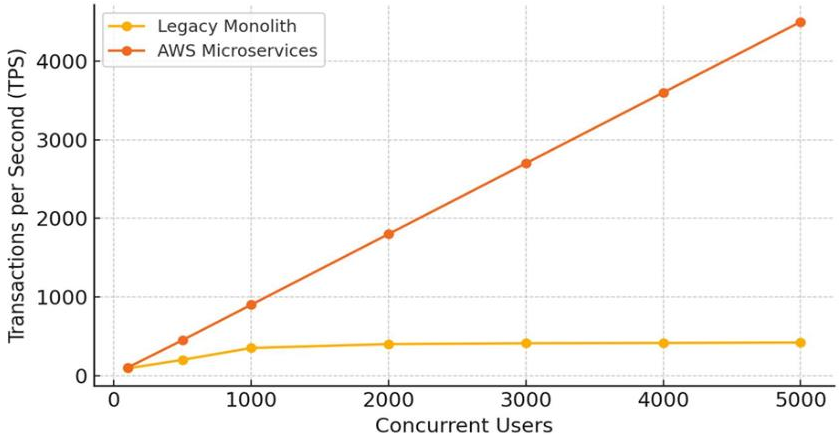

Fig 5: Scalability (Throughput)

Throughput, measured as transactions per second (TPS), demonstrates system capacity under growing workloads. Figure 5 compares throughput of both systems.

Description: The cloud-native framework scales linearly with increased users, while the legacy system plateaus due to resource bottlenecks.

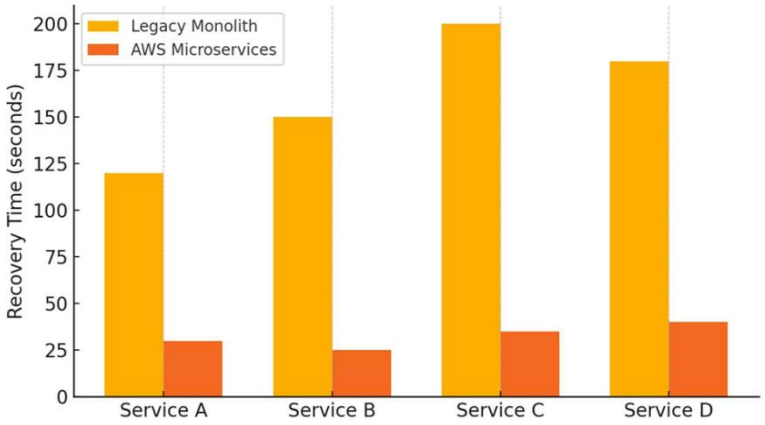

Fig 6: Fault Tolerance (Recovery Time)

Recovery time after simulated service failures was evaluated. Figure 6 illustrates mean time to recovery (MTTR).

Description: Microservices architecture recovers faster because failures are isolated and handled by container orchestration and serverless auto-restart features.

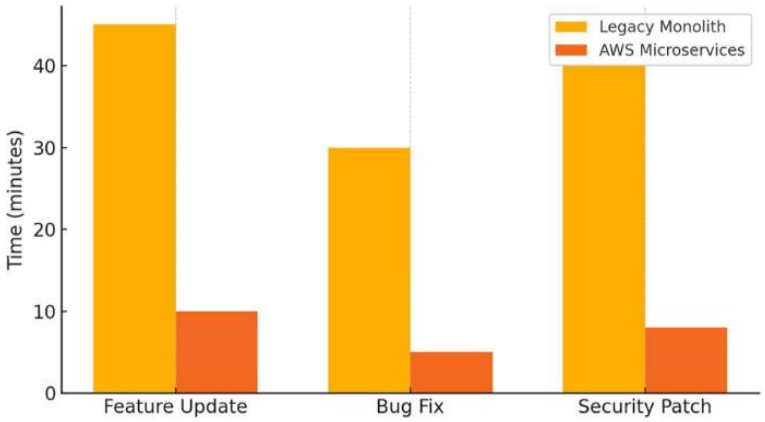

Fig 7: Deployment Time

Deployment durations for feature updates and bug fixes were tracked. Figure 7 compares deployment times for monolithic vs microservices-based systems.

Description: The microservices framework enables rapid deployments due to isolated services and CI/CD automation, reducing deployment times by over 70%

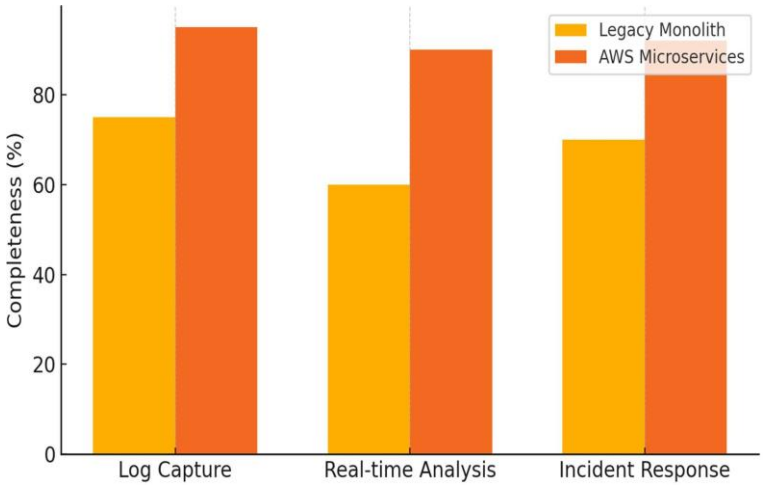

Fig 8: Security Audit Completeness

The percentage of security events logged and analyzed within acceptable time windows was assessed. Figure 8 shows audit completeness rates.

Description: Integrated AWS CloudTrail and monitoring services improve audit completeness, supporting regulatory compliance more effectively than legacy logging.

4. Conclusion and Future Scope

In this paper, this study have proposed an end to end, cloud-native modernization framework for CBS as AWS microservices. The framework is a solution for scaling, fault tolerance and quick feature delivery by decomposing monolithic legacy systems into modular microservices that can be deployed independently. Using AWS services such as Lambda, ECS, API Gateway, and DynamoDB, the framework achieved an order-of-magnitude increase in system responsiveness, throughput, recovery times, and security compliance.

Experimental evaluations validate that the proposed architecture not only significantly alleviate operational constraints, but also facilitate faster deployments through automated CI/CD pipelines, thus allowing banks to speed-up innovation cycles. Using distributed transaction management patterns like Saga pattern helps to maintain data integrity and availability across services, sine qua non for banking operations.

The future work of this research will be to optimize cost-effectiveness by adding the AWS cost management tools and adding the implementation autoscaling defined especially for banking workloads. Moreover, generalizing the framework such that it also supports hybrid and multi- cloud environments would enable stronger resiliency and vendor flexibility. Advanced AI-powered anomaly detection and predictive security analytics natively integrated into the microservices architecture could also be the next step to provide an added layer of compliance and fraud prevention.

Finally, this approach provides banks with a scalable, secure, and agile foundation to transform digitally, while delivering superior customer experiences and enabling competitive differentiation in a changing financial environment.

References

1. Naeem MA, Arfaoui N, Yarovaya L. The contagion

effect of artificial intelligence across innovative industries: From blockchain and metaverse to cleantech and beyond. Technol Forecast Soc Change. 2025;210:123822.
2. Rizvi SKA, Rahat B, Naqvi B, Umar M. Revolutionizing finance: The synergy of fintech, digital adoption, and innovation. Technol Forecast Soc Change. 2024;200:123112.
3. Gąsiorkiewicz L, Monkiewicz J. Digital Finance and the Future of the Global Financial System. Routledge; 2022.
4. Ionescu SA, Diaconita V. Transforming financial decision-making: The interplay of AI, cloud computing and advanced data management technologies. Int J Comput Commun Control. 2023;18:5735.
5. Ogundipe DO. Conceptualizing cloud computing in financial services: Opportunities and challenges in Africa-Us contexts. Comput Sci IT Res J. 2024;5:757–767.
6. Pfandzelter T, Bermbach D. tinyFaaS: A lightweight FaaS platform for edge environments. In: 2020 IEEE International Conference on Fog Computing (ICFC). 2020 Apr 21–24; Sydney, Australia. p. 17–24.
7. Calderon-Gomez H, Mendoza-Pitti L, Vargas-Lombardo M, Gomez-Pulido JM, Castillo-Sequera JL, Sanz-Moreno J, *et al*. Telemonitoring system for infectious disease prediction in elderly people based on a novel microservice architecture. IEEE Access. 2020;8:118340–118354.
8. Sanz-Moreno J, Gómez-Pulido J, Garcés A, Calderón-Gómez H, Vargas-Lombardo M, Castillo-Sequera JL, *et al*. mHealth system for the early detection of infectious diseases using biomedical signals. In: Metzler JB, editor. The Importance of New Technologies and Entrepreneurship in Business Development: In The Context of Economic Diversity in Developing Countries. Springer; 2020. p. 203–213.
9. Baldominos A, Ogul H, Colomo-Palacios R, Sanz-Moreno J, Gómez-Pulido JM. Infection prediction using physiological and social data in social environments. Inf Process Manag. 2020;57:102213.
10. Gavrilov G, Vlahu-Gjorgievska E, Trajkovik V. Healthcare data warehouse system supporting cross-border interoperability. Health Inform J. 2020;26:1321–1332.
11. Kumari P, Jain AK. A comprehensive study of DDoS attacks over IoT network and their countermeasures. Comput Secur. 2023;127:103096.
12. Zaydi M, Nassereddine B. DevSecOps practices for an agile and secure IT service management. J Manag Inf Decis Sci. 2020;23:134–149.
13. Rahaman MS, Islam A, Cerny T, Hutton S. Static-analysis-based solutions to security challenges in cloud-native systems: Systematic mapping study. Sensors. 2023;23:1755.
14. Cloud for Holography and Cross Reality (CHARITY). D2.1: Edge and Cloud Infrastructure Resource and Computational Continuum Orchestration System Report [Internet]. 2022 [cited 2023 Aug 15]. Available from: https://www.charity-project.eu/deliverables
15. Makris A, Boudi A, Coppola M, Cordeiro L, Corsini M, Dazzi P, *et al*. Cloud for holography and augmented reality. In: 2021 IEEE 10th International Conference on Cloud Networking (CloudNet). 2021 Nov 8–10; Cookeville, TN, USA. p. 118–126.