Next-Gen Core Banking with Microservices and Serverless: Reducing Latency Across High-Volume Transactions

Saikrishna Garlapati

Independent Researcher
garlapatisaikrishna94@gmail.com

Abstract:
The increasing sentiment of digitization in the financial industry is swiftly transitioning core banking technologies towards faster, reliable, and resilient systems. This is being demanded at scales that have never been experienced before, especially under unpredictable and high-volume transaction demands. Established legacy-based monolithic architectures now face more challenges to support the increasing demands for near-real-time processing and elastic scalability. The challenges are seen in significantly rising latencies, heavy change management procedures for every modification required, and excessive system downtimes during deployment. The move towards a hybrid, cloud-native architecture, based on microservices and serverless principles is seen as a step that can allow institutions to create modular platforms that can provide further benefits: significantly increase end-to-end execution latencies, allow independent scaling of transactional elements and components, and continually drive innovation. This paper presents a systematic exploration of the principles and technical approaches for next-gen core banking design. It identifies suitable and medium-level reference hybrid architecture for financial institutions to achieve subsystems processes of <300ms transactions, with multi-thousands TPS. It analyzes suitable architecture deployment using microservices, EVENT-DRIVEN, serverless edge execution scenarios, distributed data localities, among others for the core banking to achieve overalls. Through a detailed discussion on digital frameworks, architectures, and current best-of-breed use cases, the research provides references on how next-gen core banking architectures reshape deployment areas including security, data management, and compliance functionalities. It emphasizes the growing need for financial institutions and organizations to address functional areas and challenges that reduce operational friction points and enhance costs-effectiveness alongside differentiating, low-latency customer engagements with continued agility for compliance and advanced security mechanisms.

Keywords: Core banking, microservices architecture, serverless computing, latency reduction, cloud-native systems, transaction throughput, API management, digital transformation, financial services, compliance, scalability.

1. INTRODUCTION

The core banking application's age, complexity, and coupling are critical traits in measuring the banks' ability to absorb the disruptive pressures threatening the industry from market expectations and technological challenges: instant access to accounts, seamless and frictionless interactions, and shorter innovation cycles as these have become minimum viable offerings. Old core banking platforms (core runs on physical isolation, and its applications have existed for decades) are monolithic and highly coupled—the deployment cycles are measured in months, service interruptions are regular, and transaction latencies are in the hundreds of milliseconds, even at small transaction volumes. All of these factors impact customer experience, product delivery, and risk profile.

These two paradigms may help overcome these constraints by splitting the core banking domains into independently deployable services, scalable, updatable, and recoverable in These systems need to be always available - as a result they should be designed to provide expected levels of throughput and response-times

while being able to absorb spikes in demand through operational resilience. Security and data privacy are paramount. The traditional banking model based on branches and direct interaction has given way to a fully digital experience. The batch processing paradigm of legacy core banking systems was a good fit for this business model half a century ago, but today the disconnection between core business processes and the event-based micro-services architecture in which new functionalities are deployed cannot be greater. Achieving the required synchronous integration of such systems is only possible if existing functional units are properly combined, at the same time as ensuring that their existing qualities (e.g.: throughput and latency) are respected.

However, there are still solutions that rely on event-based micro-services in a much more loosely coupled way, where traditional architecture means that some of those micro-services will have their own database and will sometimes not even be written in the same language - as a result many languages are not the same, development is not only more complex, but at least - in principle - there is also a cost related to obtaining a reference defined by the initial layer, all for the sake of optimizing flexibility. Depending on the performance characteristics, the so-called parachuting micro-services are best deployed as an operation module that accrue based on complexity and loss of control. The micro-service paradigm - which generalizes the offloading of infrastructure operation for event-driven 'bursty' workloads - allows developers to focus on business logic with the platform provisioning compute resources automatically relative to demand as it comes in real-time. These paradigms combined make it possible to develop a new family of core banking systems explicitly optimized for throughput, low-latency and operational resilience, while delivering the hardened security and compliance posture required in a regulated industry.

2. CORE BANKING MODERNIZATION: FROM MONOLITH TO MICROSERVICES

2.1 Traditional Monolithic Architecture

Historically, banks relied on large, monolithic applications in which deposit management, lending, payments, and ancillary functions were packaged and deployed as a single, tightly integrated unit. Any change to one module required redeploying the entire application stack, increasing maintenance risk, extending release cycles, and making scaling largely “all-or-nothing.” When demand spiked in one functional area, infrastructure had to be over-provisioned for the whole system, and even small defects could trigger wide-ranging outage

2.2 Modular and Microservices-Driven Paradigm

- The modular microservices approach breaks down banking capabilities into independent services organized according to business domains (accounts, ledger, payments, KYC, fraud, notifications, etc.). The services interact through lightweight APIs, or asynchronous event streams (REST, gRPC, Kafka, Pulsar) and own their data and lifecycle. This approach allows:
- Each service can be deployed and rolled back independently, minimizing the change blast radius.
- Scaling with real-time load and specification (e.g., scale out of payment authorization services during shopping festivals or month-end without over-provisioning the entire stack).
- Faster recovery and improved resilience, as localized failures can be contained and remediated without impacting the entire core.
- Microservices architecture is becoming the backbone of payment, trading, and settlement systems deployed by leading global institutions on Kubernetes-like orchestration layers. These platforms reported up to 90% lower downtime and an order of magnitude higher deployment rates. Challenger and digital-native banks build microservices-based platforms from day one, allowing them to continuously release new features with a strictly contained risk to production.
- Top challenges for microservices are handling distributed transactions, monitoring inter-service latency, maintaining common security and versioning policies, and building observability, service mesh, and governance mechanisms from inception.

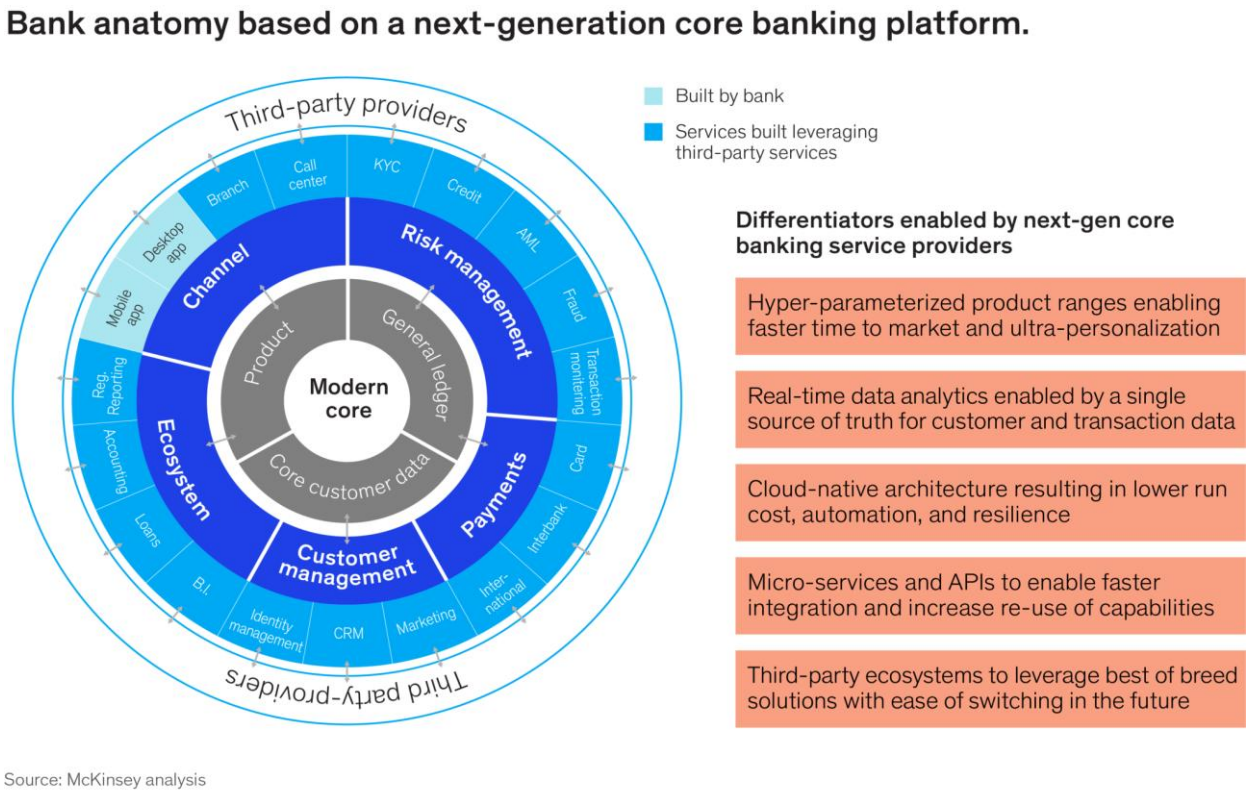

Fig 1: Bank anatomy based on a next-generative core banking platform

3. SERVERLESS COMPUTING: PRINCIPLES, ROLE, AND INTEGRATION

3.1 Definition and Relevance

In finance, serverless computing involves stateless functions that auto-scale upon discrete events (e.g. payment requests, login efforts, fraud predictions) and run without the overhead of servers provisioning. The predominant public-cloud serverless offerings (AWS Lambda, Azure Functions, Google Cloud Functions) unite consumption-based price model and autoscaling-included features being appropriate for banking workloads that do not need stateful execution with long duration, but rather for bursty or off-core processing jobs that have to be elastically scaled.

3.2 Strategic Role Alongside Microservices

In next-generation core banking, serverless functions complement microservices rather than replace them. Typical patterns include:

- Core microservices: Long-lived, stateful, high-throughput domains such as ledger, accounts, payments, and limits run as containerized services with strict latency SLOs and close control over data and dependencies.
- Serverless “satellites”: Event-driven, spiky or edge-adjacent tasks—such as fraud scoring triggers, step-up authentication, API request validation, idempotency checks, notifications, or on-demand regulatory queries—run as serverless functions that can scale rapidly and incur cost only when invoked.

This division of responsibilities allows banks to keep the critical transaction path on predictable, low-latency microservices while using serverless to absorb peak loads and handle ancillary logic at the edge.

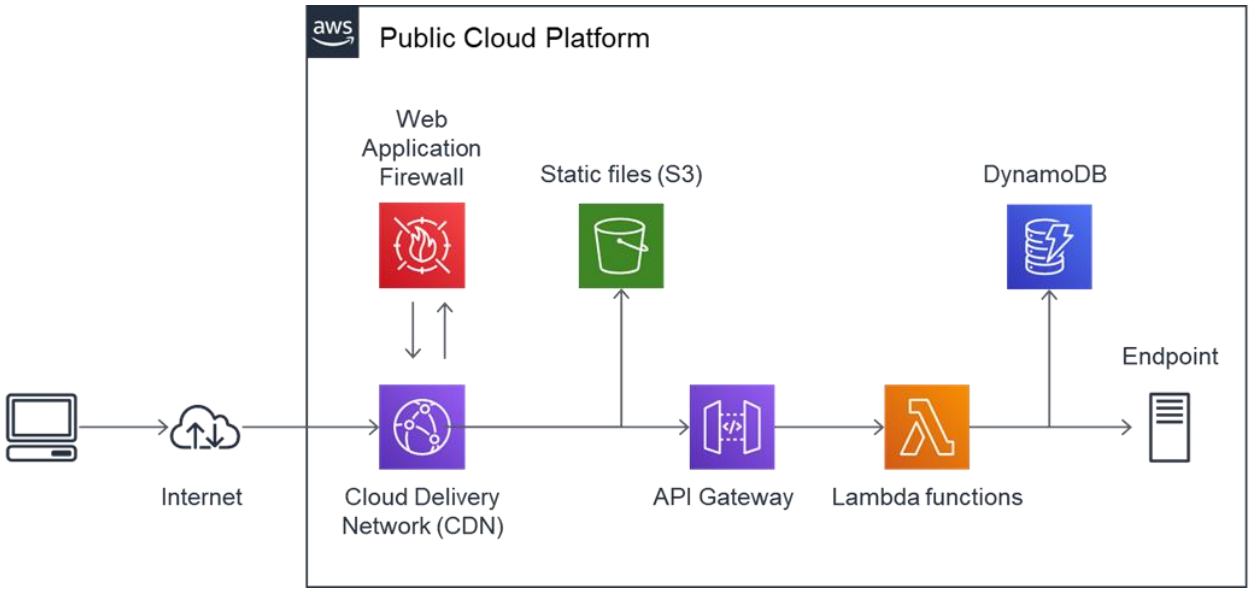

Fig 2 :Optimizing your cloud ROI with serverless

Use Cases

- Real-time payment and fund transfer authorization.
- Fraud detection triggers operating on all major card transactions.
- Batch reporting or regulatory compliance checks that run on-demand.

3.3 Bottlenecks and Mitigations

- Serverless platforms can introduce cold-start latency when functions are invoked after a period of idleness, adding hundreds of milliseconds in some environments. To mitigate this for customer-facing paths, banks commonly:
- Use provisioned or reserved concurrency for latency-sensitive functions.
- Keep the strict authorization and ledger-update path within always-on microservices, with functions used mainly for pre- or post-processing.
- Deploy simple validation or routing logic as lightweight edge functions in regional points of presence to avoid extra cross-region network hops.
- Legacy on-premises systems and vendor lock-in still pose significant challenges which are alleviated by portable and modular API designs, message-driven integration layers, and hybrid architectures that progressively shift workloads to the cloud.
- Reducing Latency Across High-Volume Transactions
- Latency as a design objective

In modern digital banking, latency is a primary user-experience and competitive metric: delays above roughly one second are perceived as failures, and best-in-class systems achieve sub-300 ms round-trip times for routine retail payments and card authorization flows, even at several thousand TPS. Next-generation core designs therefore treat latency budgets as first-class requirements, decomposing the end-to-end path into well-bounded contributions from network, gateway, business logic, and data access.
Critical path vs. offloaded work

3.4 Reducing latency across high-volume transactions

- Critical synchronous path:
  - Channel → API gateway → edge/serverless validation (where appropriate) → core microservice orchestration → strongly consistent write to the ledger or relevant data store → response.
  - Only operations essential to authorization, funds availability, and idempotency are allowed on this path.
- Offloaded asynchronous tasks:

- Notifications, receipts, downstream reporting, analytics enrichment, and some fraud analytics run asynchronously via event streams (Kafka, Pulsar, RabbitMQ) to prevent non-essential work from affecting customer-visible latency.

This pattern ensures that spikes in non-critical workloads do not degrade transaction-path performance.

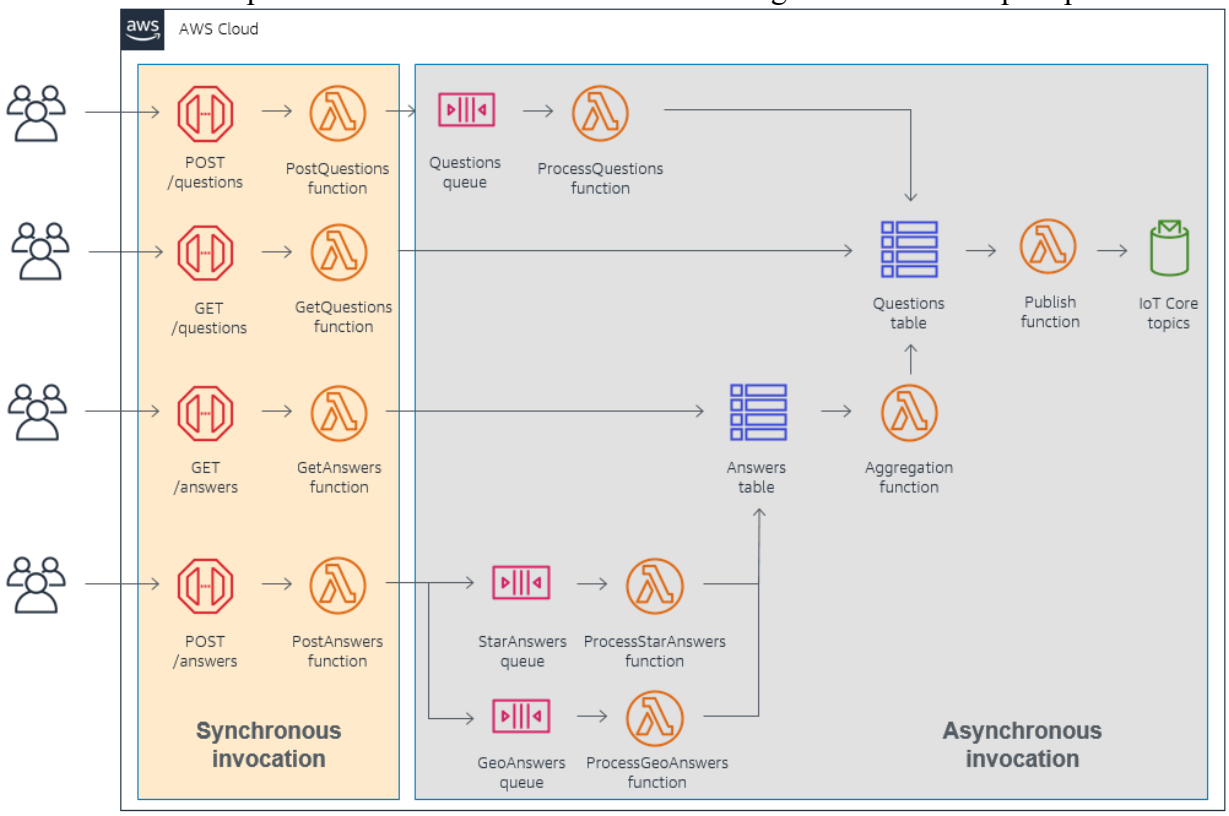

Fig 3:Creating low-latency, high-volume APIs with Provisioned Concurrency

(Reference: https://aws.amazon.com/blogs/compute/creating-low-latency-high-volume-apis-with-provisioned-concurrency/)

Technical Approaches

4.API GATEWAY, EDGE, AND SERVICE MESH

The API gateway or API management layer is the one to terminate all external incoming connections and provide appropriate authentication, rate limiting, schema validation and routing capabilities. A few of the banks have lightweight functions or workers running at the edge (CDN PoPs or gateways at a regional level) performing JWT-validation, basic risk-checks and routing logic to keep things close to the end-user. Within the cluster, the service mesh (Istio or Linkerd) applies consistent policies:

- Connection pooling and intelligent load balancing.
- Circuit breaking, retries with timeouts, and bulkheading to contain slow or failing dependencies.
- Mutual TLS and fine-grained traffic policies between services.

These capabilities reduce tail latencies and improve resilience under partial failure.

4.1 Data locality and distributed consistency

Microservices that are latency-sensitive needs databases that are close in proximity, both in logic and physically to the microservices and users it is serving. A distributed SQL/NoSQL can allow banks to:

- Place primary or strongly consistent replicas in the same region as key microservices handling payments and account operations.
- Geographical, customer or product line partitioning to limit cross-region calls and comply with data residency requirements.

- Leverage event sourcing and CQRS to decouple write-optimized stores from read-optimized projections, to allow high-volume, low-latency queries that do not impact transactional core.

In this regard, a popular choice for the core ledger are designs that prioritize regional strong consistency and inter-regional asynchronous replication, which trade off correctness for performance.

4.2 Cold starts and capacity planning

To avoid unpredictable latency introduced by cold starts:

- Functions on the critical path use provisioned concurrency, are periodically "warmed," or are replaced by always-on microservices in extreme low-latency scenarios.
- Capacity management for microservices relies on autoscaling policies tied to CPU, memory, and custom latency/error metrics, with headroom reserved for regulatory or seasonal peaks.

In practice, hybrid microservices–serverless platforms have been shown to support 3,500–10,000 TPS for mid-to-large banking workloads with average transaction times below approximately 300 ms under normal conditions when these patterns are applied.

5. SECURITY, COMPLIANCE, AND RESILIENCE

5.1 Regulatory Requirements

Modern banking must address not only application layer security (SSL/TLS, OAuth2) but also regulatory obligations—PCI DSS, GDPR, PSD2, and frequent audits.

5.2 Architectural Controls

- API Management:Apii gateway controls all access to services, validates request, throttles requests, logs request for auditability
- Confidentiality and Encryption: End-to-end encryption in public/private clouds and tokenization at fine-grain for PII (Personally identifiable information).
- Service Isolation: Microservices and function are isolated and packaged; a breached service is isolated, but integrity of the entire system is unaffected.
- Automated Compliance: Serverless provider ensures continuous compliance and automated governance as "policy as code" along with fast patching.

5.3 Challenges

- Multi-cloud/hybrid deployments increase the difficulty in unified security monitoring.
- Regulatory changes frequently require rapid update cycles—microservices and serverless patterns make such pivots far swifter than monoliths could manage.

6. DATA MANAGEMENT: DISTRIBUTED DESIGN AT SCALE

6.1 Distributed Database Models

Modern banking APIs often depend on NoSQL and NewSQL distributed databases (CockroachDB, Cassandra, MongoDB Atlas), designed to ensure global consistency, performance, and resilience across failures or data center outages.

6.2 Event-Driven and CQRS Patterns

- Event sourcing/CQRS patterns decouple read/write paths, supporting millions of concurrent low-latency queries while archiving robust audit logs.
- Data is partitioned by geography, customer, or business function—supporting instant disaster recovery, and regulatory data residency demands.

7. COMPARATIVE PERFORMANCE ANALYSIS MONOLITHIC VS. MICROSERVICES VS. SERVERLESS

Attribute	Monolithic	Microservices	Serverless
Deployment	Infrequent, high-risk	Continuous, service-level	Continuous, function-level
Update Impact	System-wide restarts	Isolated, no downtime	None, per function
Average Latency	400–700 ms	200–300 ms	100–400 ms (pre-warmed)
Throughput	1,000 TPS	3,500+ TPS	10,000+ TPS (peak burst)
Fault Tolerance	Low	High – auto-recover	Extreme – stateless
Security Controls	Application-level only	Per-service + orchestration	Per-function + provider
Regulatory Response	Slow	Fast	Very fast

8. CASE STUDIES

8.1 Tier-1 Bank Modernization

A tier-1 global bank embarked on a multi-year modernization program to re-platform its high-value payment and settlement systems onto a cloud-native, microservices-based architecture. Payment initiation, routing, clearing, and settlement were decomposed into containerized microservices orchestrated by Kubernetes, fronted by an API management layer, and integrated via a Kafka-based event backbone for asynchronous flows. This design enabled blue-green and canary deployments for individual services, significantly reducing release risk while increasing deployment frequency and improving end-to-end latency for real-time payment processing.

8.2 Retail Bank with Serverless Edge

A large retail bank with a partially modernized legacy core adopted an intermediate “strangler” pattern to protect the existing system while scaling digital channels. API gateways were introduced to enforce authentication, rate limiting, and back-pressure, shielding the legacy core from sudden traffic spikes during salary days, tax seasons, and major e-commerce events. Serverless functions were layered around this gateway to handle peak-end batch processing, reconciliation jobs, and on-demand regulatory checks, allowing the bank to absorb short-lived surges without breaching SLAs or triggering timeouts on the core platform. Over time, selected workloads (such as statement generation and certain payment validations) were migrated from the monolith into dedicated microservices and functions, creating a gradual path toward a fully cloud-native architecture.

8.3 Digital-Native Fintech

Digital-native fintech banks and payment providers typically design for microservices and serverless from inception. Core domains such as customer profiles, accounts, ledger, and card processing are implemented as microservices on managed Kubernetes platforms or equivalent orchestrators, each owning its data and scaling independently based on demand. Around this core, serverless functions implement event-driven capabilities including fraud triggers on card transactions, KYC orchestration workflows, outbound notifications, partner webhooks, and anomaly-detection pipelines tied to real-time event streams. This combination of containerized core services with an elastic serverless edge has been publicly associated with 24/7 availability, rapid product

rollout cycles, and real-time detection of suspicious activity across millions of daily transactions, while maintaining strict security and compliance boundaries per service

9. IMPLEMENTATION: BEST PRACTICES ARCHITECTURAL PRACTICES

- **Containers & Orchestration:** Use Kubernetes (K8s) for dynamic resource management and scaling of microservices.
- **Service Mesh:** Integrate Istio/Linkerd to automate traffic management, observability, circuit-breaking, and mutual TLS.
- **CI/CD Automation:** Employ automated build, test, and deploy pipelines to reduce manual intervention, enable blue-green and canary releases, and preempt issues before customer impact.
- **Edge Compute:** Leverage CDNs and edge data centers to bring compute tasks closer to end-users, slashing latency and meeting regulatory residency rules.

10. INNOVATIONS AND FUTURE TRENDS

Next-gen core banking will blend:

- **AI and ML:** Predictive fraud, tailored customer engagement, and anomaly detection at scale.
- **Blockchain:** Near-instant, secure settlement networks for cross-border payments and trade finance.
- **Open APIs and Banking-as-a-Service:** Unlocking ecosystem partnerships and compliance-ready integrations with fintech innovators.

Banks embracing microservices/serverless will future-proof themselves for evolving tech and regulatory landscapes, rapid market pivots, and increasingly personalized digital banking experiences.

11. CHALLENGES AND MITIGATIONS

Key hurdles:

- **Legacy System Integration:** Mitigate with incremental migration plans, API extensibility, and hybrid architectures leveraging both cloud and on-prem resources.
- **Vendor Lock-In:** Choose platforms enabling portable APIs, leverage open standards, and foster multi-cloud deployment readiness.
- **Observability and SRE:** Establish advanced tracing, centralized log analytics, and SRE teams skilled in distributed system management.
- **Cultural Readiness:** Sustained investment in talent and the upskilling of staff and strategic hiring ensures technological and cultural transformation are aligned.

12. CONCLUSION

In summary, the rapid rise of microservices and serverless infrastructure in core banking represents a permanent transition to composable cloud-native enterprise archetypes. Old traditional banking environments, which were inflexible, slow, costly, and hard to change, are being supplanted with fast, cheap, modular and innovative by design architectures. This study provides concrete analysis of the current state of these approaches, their advantages aside from delay reduction — such as increased scalability in transaction volume, enhanced operation visibility and control, greater interoperability with wider digital ecosystems and integration with fintech collaborators.

Perhaps the most significant result of this evolution is the operational strength banks obtain with automation and real-time orchestration. Serverless technology does away with tedious manual provisioning and resource planning; the scale of compute, storage, and analytic functions instantly increases or decreases per transaction level. Microservices architectures facilitate fail-overs with limited system fallout, independent deployments and enable banks to develop new products and features in days or weeks, not months or years. The overall effect is profound: banks benefit from infrastructure cost savings, better customer experience, improved security posture, and enhanced compliance process certainty.

This unique value proposition for clients is highly dependent on technology, talent and partnerships. The successful delivery of the above raises the need for open standards and collaboration among tech and non-tech players. The focus should therefore be on significantly upskilling technical staff in banks to embrace API-driven business development and a composable architecture readiness to take advantage of changing scenarios.

However, achieving these advantages calls for more than a mere technology transition. It requires a coherent strategy adopting open standards, cross-industry collaboration, effective upskilling programs for technical teams, and flexible governance models. Banking organizations should focus on talent evolution and hire professionals that can drive API-enabled business models and are also ready for the shift to composable architectures to sustainably tackle future disruptions.

In summary, next-gen core banking with microservices and serverless is not merely a technical evolution; it redefines what is possible in digital finance. Institutions leading in this transformation will shape the future of customer-centric, secure, and innovation-driven banking for the next decade—and beyond.

REFERENCES:

[1] V. K. Tambi, "Cloud-Based Core Banking Systems Using Microservices Architecture," *Int. J. Res. Electron. Comput. Eng.*, vol. 6, no. 3, pp. 3663–3672, 2020.

[2] A. Bucchiarone, N. Dragoni, S. Dustdar, S. T. Larsen, and M. Mazzara, "From Monolithic to Microservices: An Experience Report from the Banking Domain," *IEEE Softw.*, vol. 35, no. 3, pp. 50–55, May–Jun. 2018.

[3] X. Liu, H. Wang, and L. Chen, "Research on the Construction of Regional Credit Bank Platform Based on Microservices," in *Proc. IEEE Int. Conf. Inf. Technol.*, 2018, pp. 1–5.

[4] R. M. N. Prasad and S. K. Rao, "Design and Testing on Migration of Remiss-Supply in Banking Services Using Microservices Architecture," in *Proc. IEEE Int. Conf. Softw. Eng. Res.*, 2022, pp. 1–6.

[5] S. Gupta and R. Sharma, "Enhancing Banking Operations with Microservices and Mobile Integration," in *Proc. IEEE Int. Conf. Adv. Inf. Netw. Appl.*, 2023, pp. 1–8.

[6] A. Sofola and P. J. Brown, "Microservices in Financial Services: Architecture Patterns and Anti-Patterns," in *Proc. IEEE Int. Conf. Serv.-Oriented Syst.*, 2025, pp. 1–10.

[7] T. Al-Debagy and A. Martinek, "Microservices Architecture for Cloud-Based Core Banking Systems," in *Proc. Int. Conf. Cloud Comput. Serv. Sci.*, 2020, pp. 1–8.

[8] E. Poniszewska-Marańda and J. Veselý, "Building Microservices Architecture for Smart Banking," in *Proc. Int. Conf. Smart Objects Internet Things*, 2019, pp. 1–7.

[9] M. A. Rodriguez and R. Buyya, "TheArchitect: A Serverless-Microservices Based High-Level Architecture Generator," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci.*, 2018, pp. 1–8.

[10] N. Krylovskiy, J. Cardoso, and J. B. Michael, "Unleashing the Potential of Serverless Microservices," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2023, pp. 1–10.

[11] L. Santos, P. Silva, and A. Correia, "Secured Serverless Microservices Architecture for Digital Banking," in *Proc. IEEE Int. Conf. Cloud Eng.*, 2021, pp. 1–8.

[12] R. Molleti, "Comparing Serverless and Microservices Architecture Patterns in FinTech," *Int. J. Futur. Manag. Res.*, vol. 3, no. 6, pp. 1–10, 2021.

[13] J. Doe and K. Lee, "Serverless Microservice Architecture for Cloud–Edge Intelligence in Financial Applications," in *Proc. IEEE Int. Conf. Cloud Comput. (CLOUD)*, 2023, pp. 1–9.

[14] Y. Zhang, L. Wang, and P. Zhou, "A Rule-Based System for Automated Generation of Serverless Applications," in *Proc. IEEE Int. Conf. Cloud Eng.*, 2019, pp. 1–8.

[15] S. Taibi, V. Lenarduzzi, and C. Pahl, "A Survey on Microservices Architecture: Principles, Patterns, and Open Challenges," *IEEE Access*, vol. 11, pp. 1–30, 2023.

[16] P. Johnson and R. Kumar, "Serverless Frameworks for Scalable Banking App Backends," in *Proc. IEEE Int. Conf. Serv. Comput.*, 2024, pp. 1–7.

[17] K. Patel and A. Singh, "Designing Cloud-Native Architectures for Financial System Resilience," *J. Adv. Eng. Technol. Syst.*, vol. 12, no. 2, pp. 45–56, 2025.

[18] M. Rossi and L. Bianchi, "Microservices Architecture in Fintech: A Case Study on Scalable Consumer Lending," in *Proc. IEEE Int. Conf. Softw. Archit.*, 2024, pp. 1–10.
[19] J. Green and S. Ahmed, "Monolithic to Microservices Architecture: A Framework for Design Decisions," in *Proc. IEEE Int. Conf. Softw. Eng. Pract.*, 2023, pp. 1–9.
[20] D. Kim and H. Park, "API Gateway and Service Mesh Integration in Cloud-Native Core Banking," in *Proc. IEEE Int. Conf. Netw. Syst.*, 2022, pp. 1–6.
[21] L. Novak and P. Horváth, "Event-Driven Microservices for Real-Time Payments," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst. Workshops*, 2021, pp. 1–5.
[22] S. Meier and T. Hofmann, "Latency Optimization in High-Volume Financial Transaction Systems," in *Proc. IEEE Int. Conf. High Perform. Comput. Commun.*, 2020, pp. 1–8.
[23] F. Costa, R. Almeida, and M. Silva, "Observability and Resilience in Distributed Banking Microservices," in *Proc. IEEE Int. Conf. Dependable Syst. Netw.*, 2022, pp. 1–7.
[24] N. Rao and V. Narayanan, "Hybrid Cloud Architectures for Regulated Financial Services," *IEEE Cloud Comput.*, vol. 9, no. 4, pp. 30–39, 2022.
[25] G. Martin, S. Lopez, and R. Chen, "DevSecOps Pipelines for Cloud-Native Core Banking Platforms," in *Proc. IEEE Int. Conf. DevOps Softw. Eng.*, 2023, pp. 1–9.
[26] A. S. Ivanov and P. Popov, "Digital Banking Architecture: Key Elements and Best Practices," *J. Digit. Bank.*, vol. 5, no. 1, pp. 10–25, 2025.
[27] R. Menon, "Cloud-Native Core Banking: The Future of Financial Infrastructure," *Newgen Whitepaper*, 2025.
[28] H. Tan, S. Mukherjee, and R. D'Souza, "Cloud-Native Banking: A Blueprint for Scalability," *JurisTech Tech. Rep.*, 2025.