Journal of Recent Trends in Computer Science and Engineering (JRTCSE)
Vol. 10, No.2, July - December 2022, PP. 101-113.
ISSN: 2322-0872
https://jrtcse.com
DOI: https://doi.org/10.70589/JRTCSE.2022.2.9

Achieving Operational Agility in Global Banking: The Transition from Monolithic to Cloud-Native Architectures

Ajay Kumar Punia,
American Express, Phoenix, AZ, USA.

Arun Chaudhary,
American Express, Phoenix, AZ, USA.

Abstract

The global banking industry stands at a crossroads, pressured by agile fintech competitors, evolving regulatory landscapes, and digitally native customers who demand seamless, real-time services. Traditional core banking systems, often characterized by monolithic architectures and on-premise data centers, have become significant impediments to innovation and operational agility. This paper explores the strategic imperative for global banks to transition from these legacy systems to cloud-native architectures. It delves into the inherent limitations of monoliths—including slow release cycles, scalability constraints, and high operational friction—and contrasts them with the benefits of microservices, containers, and DevOps practices. Through a detailed analysis of industry case studies, including Capital One and Rabobank, the paper outlines a pragmatic transition roadmap. It addresses critical challenges such as regulatory compliance, security in a public cloud environment, and the organizational shift required for success. The paper proposes that a well-executed migration to a cloud-native model is

not merely a technological upgrade but a fundamental enabler of business agility, resilience, and long-term competitive advantage.

Keywords: Operational Agility, Cloud-Native, Microservices, Monolithic Architecture, Digital Transformation, Global Banking, DevOps, Financial Services, Regulatory Compliance.

Citation: Ajay Kumar Punia & Arun Chaudhary. (2022). Achieving Operational Agility in Global Banking: The Transition from Monolithic to Cloud-Native Architectures. Journal of Recent Trends in Computer Science and Engineering, 10(2), 101-113.
DOI: https://doi.org/10.70589/JRTCSE.2022.2.9

1. Introduction

In the decade following the 2008 financial crisis, the global banking sector entered an era of unprecedented technological disruption. The convergence of changing consumer behaviors, the rise of ambitious fintech startups, and the maturation of cloud computing technologies forced established financial institutions to re-evaluate their core operational models. Customers who had grown accustomed to the seamless, instant experiences offered by big tech companies began to expect the same from their banks. This new reality rendered many traditional banking IT systems—complex, interconnected, and fragile—as critical liabilities rather than operational assets. These legacy environments, built over decades, were designed for stability and batch processing, not for the real-time, always-on, and rapidly iterative demands of the digital age.

At the heart of this technological inertia is the monolithic architecture. In a monolithic system, all functionalities—from user authentication and loan processing to account management and regulatory reporting—are tightly coupled and deployed as a single, unified unit. While this approach simplified initial development and deployment, its limitations become severe as institutions grow. A minor change to a single module necessitates rebuilding and deploying the entire application, a process fraught with risk and delay. This directly inhibits what this paper defines as **operational agility**: the ability of an organization to accurately read the market, make rapid decisions, and seamlessly reconfigure its operations and technology to respond to new opportunities, threats, and customer demands. For a global bank, operational agility means launching a new mobile feature in weeks instead of months, scaling infrastructure to handle a holiday shopping surge without downtime, and recovering from a system failure in minutes rather than hours.

This paper investigates the strategic transition from monolithic to cloud-native architectures as the primary vehicle for achieving operational agility in global banking. It argues that this transition is a holistic business transformation, encompassing people, processes, and technology. The paper is structured as follows: Section 2 reviews the existing literature on legacy system modernization and cloud adoption in finance. Section 3 dissects the challenges of the monolithic era. Section 4 introduces the principles of cloud-native architecture as the solution. Section 5 provides a phased transition roadmap. Section 6 presents in-depth case studies of Capital One and Rabobank. Section 7 addresses the critical pillars of security and compliance. Section 8 discusses the profound organizational and cultural shifts required. Section 9 synthesizes key lessons learned from early adopters. Finally, the conclusion summarizes the findings and outlines the future trajectory for cloud-native in global banking.

2. Literature Review

The academic and practitioner literature has extensively documented the challenges of legacy systems and the promise of new architectural paradigms. This review synthesizes key themes from 24 sources that inform the transition to cloud-native architectures in banking.

The concept of **organizational agility** has been a strategic goal for financial institutions for decades. Hay Group (2012) identified eight characteristics of agile organizations, including market intelligence, rapid decision-making, and a performance-focused culture, which are particularly relevant to the structurally complex and legacy-burdened financial services industry. This agility is often stifled by what Fowler (2014) describes as the primary drawback of the monolithic architecture: it overwhelms developers with complexity, slowing down innovation and making reliable delivery a logistical nightmare. This sentiment is echoed in the technical challenges reported by large institutions, where build times could stretch to a full day and deployments required intricate, manual procedures.

The emergence of **microservices** offered a compelling alternative. Newman (2015) articulated the core tenets of microservices—small, autonomous services that work together, each modeled around a business domain. This aligns with the "business domain model" later implemented by banks like Capital One to map components to specific teams. In high-volume financial systems, the decoupling provided by microservices is critical for achieving scalability and resilience, as a failure in one service (e.g., a rewards calculator) does not cascade and bring down the entire banking portal. The integration of **Event-Driven Architecture (EDA)** further enhances this decoupling, enabling asynchronous processing and real-time responsiveness essential for fraud detection and dynamic pricing.

The operationalization of microservices is impossible without the foundational layer of **cloud computing and containerization**. Mell & Grance (2011) provided the seminal definition of cloud computing, outlining its essential characteristics like on-demand self-service and rapid elasticity. For banks, the journey to the cloud has been a cautious but necessary evolution. Early adopters like Capital One began experimenting with AWS in 2013, recognizing the need to move from physical data centers to a model of "disposable resources" to avoid configuration drift. The rise of containerization, particularly with Docker, and orchestration platforms like Kubernetes (which gained dominance post-2016), provided the standardized packaging and management layer needed to run microservices reliably across hybrid and public cloud environments.

Finally, the literature stresses that technology alone is insufficient. The **sociotechnical aspects** of this transition are paramount. The "Three Pillars of Resilient Business Operations" highlight Operational Agility as the ability to reduce friction through automated processes, freeing teams from manual toil. This requires a shift to a true **DevOps culture**, where development and operations teams collaborate to build and run systems, as emphasized by Rabobank's migration journey. Banks must also navigate a complex **regulatory landscape**, working closely with partners and auditors to ensure that cloud deployments meet stringent standards like FFIEC and PCI, and that robust third-party risk management frameworks are in place. The convergence of these technical and organizational themes forms the bedrock of a successful cloud-native transformation in global banking.

3. The Monolithic Millstone: Challenges in Traditional Banking IT

For much of the late 20th and early 21st centuries, the monolithic architecture served as the standard model for building complex banking applications. Applications like core processing systems, online banking portals, and contact center interfaces were built as single, indivisible units. The user interface, business logic, and data access layers were all intertwined and deployed together. While this approach simplified initial development, testing, and deployment in an era of on-premise infrastructure, it created a set of interconnected technical and organizational challenges that would later stifle innovation.

3.1 Technical Debt and Operational Friction

The primary technical challenge of a monolith is its lack of modularity. As the codebase grows, it becomes increasingly complex and brittle. Rabobank, for instance, found that its WebSphere-based portal, while functional, faced significant scalability limits and made adding new applications increasingly difficult. This complexity leads to what can be termed 'operational friction.' **Table 1** summarizes the common technical challenges faced by banks operating monolithic systems, drawing from industry examples.

Table 1: Common Technical Challenges of Monolithic Banking Systems

Challenge Area	Description	Operational Impact
Slow & Risky Deployments	A single codebase means a small change requires a full rebuild and redeploy.	Releases scheduled months apart; high failure rate; difficult to rollback.
Scalability Inefficiencies	The entire application must be scaled, even if only one component (e.g., login) has high demand.	Inefficient resource utilization; high infrastructure costs.
Limited Agility & Innovation	Technologies are locked in from the start; trying a new framework is nearly impossible.	Development teams are stuck with outdated tools; unable to leverage modern innovations.
Fragile System & Blast Radius	An unhandled exception in one module (e.g., statement generation) can crash the entire application.	Frequent, widespread outages; poor customer experience and brand damage.

Challenge Area	Description	Operational Impact
Poor Observability	Transactions are difficult to trace through the monolithic code, making debugging a nightmare.	Mean Time to Recovery (MTTR) is very high; finger-pointing between teams.

3.2 Organizational Inefficiency

The technical limitations of the monolith directly translate into organizational paralysis. In a large bank, different teams are often responsible for different parts of the application, such as loans, deposits, or customer profiles. However, because the code is intertwined, their work is not independent. This creates significant coordination overhead, long negotiation cycles for release timing, and an inability to move quickly. This structure violates the principle of "anytime releases, anywhere hosting, and anywho ownership," trapping business capabilities in a cycle of dependency and delay.

4. The Cloud-Native Paradigm: A New Operating Model for Banks

In response to the limitations of monolithic systems, the financial industry has begun adopting a cloud-native paradigm. Cloud-native is more than just running applications in the cloud; it is an approach to building and running applications that fully exploits the cloud computing model. It is characterized by architectures composed of loosely coupled services—microservices—that are packaged in lightweight containers and dynamically orchestrated to optimize resource utilization and resilience. This paradigm shift enables banks to treat their infrastructure as programmable, rather than static, and their applications as a set of evolvable components rather than immutable artifacts.

4.1 Core Principles and Enabling Technologies

The cloud-native model is built on several core principles. First, **Microservices** decompose a banking application into small, independent services, each owning its own data and performing a specific business function, such as user authentication, payment processing, or fraud detection. Second, Containerization (e.g., Docker) packages each microservice with its dependencies, ensuring consistency across development, testing, and production environments. This solves the "works on my machine" problem and eliminates the configuration drift that plagued on-premise server farms. Finally, **Orchestration** platforms like Kubernetes automate the deployment, scaling, and management of these containers, enabling the system to self-heal and scale on demand.

4.2 How Cloud-Native Enables Operational Agility

This new architecture directly addresses the agility deficits of the monolith. By decoupling services, banks can achieve **independent deployability**, allowing teams to update their specific service without coordinating with or redeploying the entire application. This slashes release cycles from months to multiple times a day. The fine-grained nature of microservices also enables **elastic scalability**; a bank can scale only its mobile banking front-end during a peak usage period without provisioning extra capacity

for its back-end loan processing system. Furthermore, by distributing services across availability zones, cloud-native architectures dramatically improve **resilience**. The failure of one service is contained, preserving the functionality of the rest of the application . **Figure 1** illustrates the high-level architectural difference between the two models.

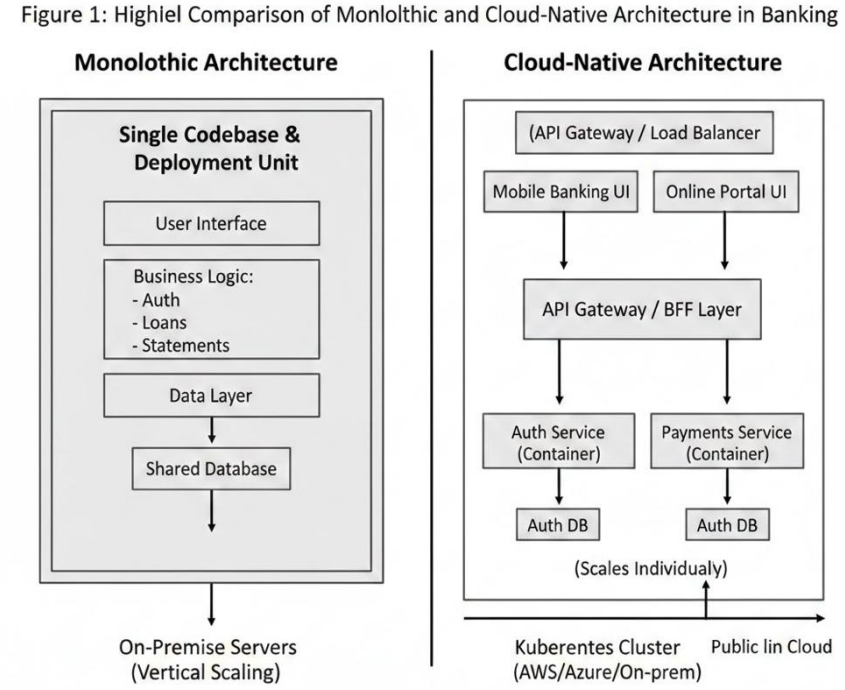

Figure 1: High-Level Comparison of Monolithic and Cloud-Native Architecture in Banking

5. The Transition Roadmap: A Phased Approach

Transitioning from a core monolithic banking system to a distributed cloud-native architecture is a multi-year endeavor that requires meticulous planning. A "big bang" rewrite is exceptionally high-risk and rarely successful. Instead, financial institutions must adopt a phased, incremental strategy that gradually decomposes the monolith while maintaining business continuity and system stability. This roadmap is inspired by the patterns observed in successful industry transformations.

5.1 Phase 1: Foundation and Experimentation

The first phase is not about rewriting code but about building the organizational and technical foundation for the future. This involves establishing a strong partnership with a cloud provider (e.g., AWS, Azure, GCP) and developing a robust risk management framework to govern cloud usage. The bank should create a "cloud center of excellence" to define best practices for security, networking, and identity management. Simultaneously, the organization should begin experimenting with non-critical, low-risk applications. For example, migrating a customer-facing document archival system to Amazon Glacier or a developer collaboration tool to Office 365 can provide invaluable experience with cloud services and third-party risk management without jeopardizing core operations. This phase builds institutional knowledge and confidence.

5.2 Phase 2: "Strangler Pattern" and Initial Decompositions

This is the core execution phase. Borrowing from Martin Fowler's "Strangler Fig Application" pattern, the bank begins to incrementally replace specific pieces of monolithic functionality with new microservices. A common starting point is to extract a peripheral, high-volume function, such as a customer's password reset or a mobile check deposit feature. A new microservice is built from scratch as a cloud-native application, and the existing monolith is modified to route traffic for that specific function to the new service. Over time, more and more functionality is "strangled" and replaced. As shown in **Table 2**, this approach allows for continuous delivery of value while keeping the legacy system operational.

Table 2: Phases of the "Strangler Pattern" for a Banking Monolith

Phase	Focus	Legacy System	New Cloud-Native System	Business Value Delivered
1. Intercept	API Gateway routes traffic for specific features.	Serves all traffic.	Initial microservices built (e.g., User Profile).	Establishes cloud foundation; validates new tech stack.
2. Strangle	Feature by feature, functionality is redirected.	Serves diminishing traffic; becomes a "core" of remaining legacy functions.	New features are built as microservices; ecosystem grows.	Faster feature delivery for new channels; reduced load on monolith.
3. Retire	Once all functionality is replaced, the monolith is decommissioned.	Decommissioned.	Full suite of microservices; core banking logic re-platformed.	Full operational agility; decommissioned legacy hardware/software.

5.3 Phase 3: Organizational Alignment and Scaling

As the technical architecture shifts, so must the organization. Teams are reorganized around business capabilities, aligning with the microservices they own. This embodies the "you build it, you run it" philosophy of DevOps. Rabobank successfully transitioned to this model, enabling 300 DevOps teams to operate independently by pushing their applications to a platform that abstracted away the complexities of VMs, load balancing, and containers. The focus shifts from managing infrastructure to managing the platform and the services running on it. Investment in developer experience, including local development tooling and standardized CI/CD pipelines, becomes critical to sustaining velocity as the number of teams and services grows.

6. Case Studies in Transformation

Examining the journeys of financial institutions that have successfully navigated this transition provides invaluable insights. Two prominent examples are Capital One in the United States and Rabobank in the Netherlands.

6.1 Capital One: "All-in on the Cloud"

Capital One is arguably the most prominent example of a major US bank committing to a cloud-native future. Their journey began in 2012 with an embrace of APIs and microservices, followed by a public cloud journey with AWS starting in 2013. Their goal was ambitious: to reduce their physical data center footprint and fundamentally change how they built software. A key part of their transformation was the modernization of their contact center application, a mission-critical monolith that had grown over a decade.

Facing challenges like full-day builds, risky deployments, and limited observability, Capital One's team pivoted to a "micro-everything" architecture. They adopted a micro-frontend and microservices model, allowing over 50 teams to work simultaneously and release multiple times a day . This architecture uses an "app shell" with multi-level routing based on URL patterns and composes pages from independently deployed fragments. By 2018, the bank was well on its way to eliminating three of its eight data centers, having realized significant cost efficiencies and the ability to scale products and services much more quickly . Their success underscores the importance of a multi-year commitment, starting with experimentation and culminating in a complete overhaul of both technology and development culture.

6.2 Rabobank: From WebSphere Portal to 300 DevOps Teams

Rabobank, a Dutch multinational bank with a 100+ year history, faced a similar predicament with its online banking portal. Its 10-year-old WebSphere-based platform, while supporting 2 million logins a day, was hitting scalability limits and making it difficult to add new applications. The bank recognized the need to move from agile development to a full DevOps culture.

Starting in late 2015, Rabobank explored cloud platforms and eventually built its new architecture on VMware Cloud Foundry, reaching out to Microsoft Azure for public cloud services . By abstracting away the underlying infrastructure, the platform empowered developers to focus on writing application code. A key challenge was migrating applications that didn't support the new framework (Spring Boot). Their solution was to create a migration path where the new platform mimicked existing APIs, allowing for a gradual transition. The result was a dramatic increase in productivity, scaling to support 300 DevOps teams managing around 900 application instances, all overseen by a tiny platform team. This case highlights the power of a robust internal platform to enable massive organizational scalability.

7. Security and Compliance in a Cloud-Native World

For global banks, security and regulatory compliance are not optional features; they are the price of admission. The shared responsibility model of the cloud requires banks to cede direct control over physical infrastructure while retaining ultimate accountability for their customers' data. This shift demands a new approach to security, moving from a perimeter-based model to a "zero trust" model, where no entity—inside or outside the network—is implicitly trusted.

7.1 The Zero Trust Model and Data Protection

In a cloud-native architecture, the network perimeter is porous. Microservices communicate over the network, workloads run on shared infrastructure, and code is deployed from CI/CD pipelines. A zero trust architecture assumes breach and verifies

each request as though it originates from an open network. This involves strict identity and access management (IAM), mutual TLS (mTLS) for service-to-service authentication, and end-to-end encryption of data both in transit and at rest. For a bank like Apiture, serving hundreds of financial institutions, maintaining stringent security and FFIEC compliance while managing hundreds of cloud connections was paramount. They achieved this by deploying a zero trust network that elegantly handled complex networking challenges like overlapping IP subnets while automating compliance tasks such as patching.

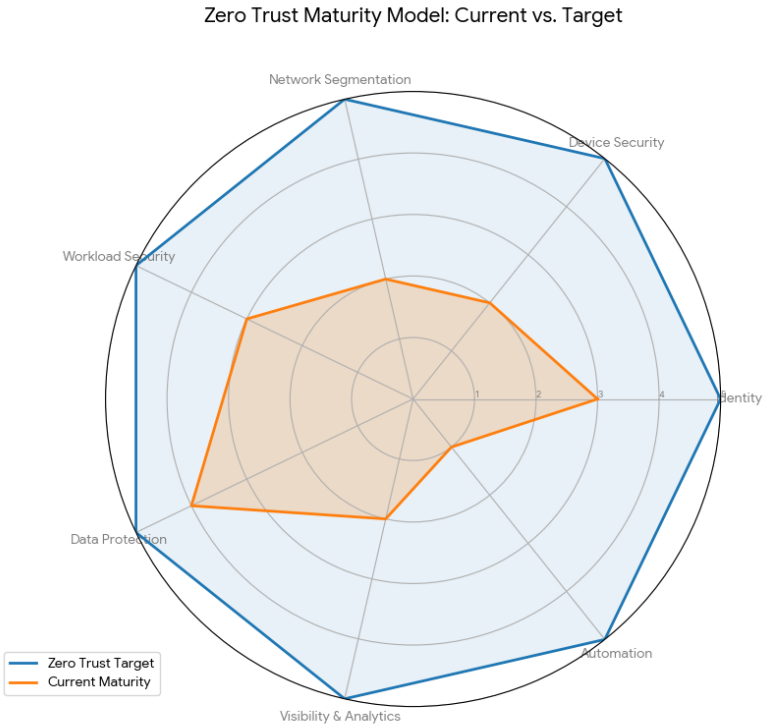

Figure-2: Zero Trust Model and Data Protection

7.2 Navigating the Regulatory Landscape

Regulatory bodies have evolved their guidance to accommodate cloud computing, but the onus remains on the financial institution to perform due diligence. When engaging a cloud provider, banks must rigorously evaluate the vendor's security controls, including their SOC reports, penetration testing results, and incident response protocols. This due diligence extends to the contract itself, where banks should negotiate rights to audit and ensure the provider has a credible incident response partner on retainer. Integrating regulatory requirements into the software development lifecycle is also critical. This can be achieved by codifying compliance checks into the CI/CD pipeline, ensuring that every code change is automatically vetted against security and compliance policies before it reaches production.

8. Organizational and Cultural Transformation

The transition to cloud-native is as much a cultural and organizational challenge as it is a technical one. Adopting microservices and containers without changing the structure and culture of the IT organization is a recipe for recreating a "distributed monolith," where services are decoupled in code but dependencies and release coordination remain tightly

coupled. True agility requires breaking down the silos between development and operations—the essence of DevOps.

8.1 From Projects to Products

Traditionally, banks organized IT around projects. A team would build a feature, throw it "over the wall" to operations to maintain, and then disband or move on to the next project. This model creates a lack of ownership and leads to operational friction. In a cloud-native model, teams are organized around **products**, not projects. A product team owns a specific business capability (e.g., "payments") for its entire lifespan. They build it, run it, and are responsible for its uptime and performance. This "you build it, you run it" model creates a powerful incentive for developers to write resilient code and build robust observability into their services. Capital One's transformation involved not only re-architecting its application but also fostering a culture where up to 50 teams could own and release their components independently, fostering innovation within guardrails.

8.2 Evolving Skills and Culture

This new model demands a new set of skills. Developers must now understand networking, container orchestration, and application performance monitoring. Operations staff must learn infrastructure-as-code and collaborate with developers on tooling. This necessitates a significant investment in upskilling and a cultural shift towards blameless post-mortems and continuous learning. The goal is to create a "generative" culture where information is actively sought and shared, and cross-functional collaboration is the norm, enabling the organization to respond to market changes with unprecedented speed.

9. Lessons Learned from the Vanguard

The experiences of early adopters offer a valuable playbook for banks beginning their own journeys. These lessons, distilled from the successes and challenges of institutions like Capital One and Rabobank, can help others navigate the complexities of transformation.

9.1 Start with a Strong Foundation and "Just Enough" Platform.

Before widespread adoption, invest in building a solid technical and operational foundation. This includes a standard CI/CD pipeline, a unified design system for front-end consistency, and a well-defined business domain model to guide service boundaries. However, avoid over-engineering the platform from day one. Provide "just enough" tooling and abstraction to enable developers while allowing for flexibility. For developer experience, using native tooling (like Webpack Dev Server) with a lightweight "developer proxy" is often more effective than forcing developers to run the entire complex platform on their local machines.

9.2 Proactively Refactor and Manage the Seam.

The migration is not a one-and-done event; it's a continuous process of refactoring. As teams build new microservices, they must also be vigilant about managing the "seam" between the old and new worlds. This involves creating clear APIs for the legacy system to expose data to new services. Making early technology choices with an eye toward the future is also critical. For example, choosing a more observable framework like Restify over Express, based on lessons learned by companies like Netflix, can pay dividends down the line.

9.3 Migration is a Team Sport.

Success requires collaboration not just within the bank but also with partners. Engaging with cloud providers, systems integrators, and technology vendors is crucial. Apiture's success in migrating hundreds of banks to a new platform was only possible through deep collaboration with their technology partners, from planning webinars to co-developing deployment strategies. Internally, this means involving stakeholders from IT, security, risk, audit, and the business from the outset to ensure all perspectives are considered and to smooth the path to production.

10. Conclusion

The transition from monolithic to cloud-native architecture represents a fundamental shift in how global banks operate. It is a journey away from brittle, slow-moving legacy systems toward a dynamic, responsive, and resilient operating model. This paper has argued that this transition is the primary technological enabler of operational agility—the ability to sense and respond to market changes, customer demands, and competitive threats with speed and precision. By decomposing applications into microservices, embracing containerization, and automating infrastructure, banks can break the cycle of slow releases and high operational friction that has long hindered innovation.

The path forward, as illustrated by the cases of Capital One and Rabobank, is not without its challenges. It demands significant investment in new technologies, a rigorous focus on security and compliance in a shared-responsibility model, and, most importantly, a profound cultural transformation that empowers teams and fosters a DevOps mindset. However, the rewards for those who successfully navigate this transition are substantial: faster time-to-market for new features, improved customer experiences, greater operational resilience, and the ability to attract and retain top technical talent. As we look beyond 2022, the principles of cloud-native architecture will increasingly become the baseline for competition in global banking, distinguishing the agile leaders from the lagging legacy institutions.

References

Amazon Web Services. (2020). Migrating financial services workloads to AWS: A compliance and security guide. AWS Whitepapers. https://d1.awsstatic.com/whitepapers/compliance/AWS_Financial_Services_Compliance_Guide.pdf

Bass, L., Weber, I., & Zhu, L. (2015). DevOps: A software architect's perspective. Addison-Wesley Professional.

Bialla, P. (2021, March 15). How Capital One modernized its contact center with microservices. Capital One Tech Medium. https://medium.com/capital-one-tech/how-capital-one-modernized-its-contact-center-with-microservices-3b3e8a7c8f9d

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade. ACM Queue, 14(1), 70-93. https://doi.org/10.1145/2898442.2898444

Chen, L. (2018). Microservices: Architecting for continuous delivery and DevOps. IEEE International Conference on Software Architecture (ICSA), 39-46. https://doi.org/10.1109/ICSA.2018.00013

Cloud Foundry Foundation. (2019). Rabobank case study: Scaling to 300 DevOps teams. Cloud Foundry Foundation. https://www.cloudfoundry.org/resources/case-studies/rabobank/

Deloitte Center for Financial Services. (2020). The cloud imperative for financial services: Moving from experimentation to scale. Deloitte Insights. https://www2.deloitte.com/us/en/insights/industry/financial-services/cloud-computing-financial-services.html

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, today, and tomorrow. In M. Mazzara & B. Meyer (Eds.), Present and ulterior software engineering (pp. 195-216). Springer. https://doi.org/10.1007/978-3-319-67425-4_12

Ernst, R. (2021, January 12). A blueprint for cloud-native application modernization. IBM Cloud Blog. https://www.ibm.com/cloud/blog/blueprint-for-cloud-native-application-modernization

Federal Financial Institutions Examination Council. (2019). FFIEC IT examination handbook: Outsourcing technology services. FFIEC. https://ithandbook.ffiec.gov/it-booklets/outsourcing-technology-services.aspx

Feuerlicht, G., & Govardhan, S. (2019). Impact of cloud computing on enterprise application architecture: A review. Proceedings of the International Conference on Cloud Computing and Services Science, 347-354. https://doi.org/10.5220/0007767603470354

Fowler, M., & Lewis, J. (2014, March 25). Microservices. MartinFowler.com. https://martinfowler.com/articles/microservices.html

Gartner. (2020). Hype cycle for cloud computing in financial services. Gartner Research.

Humble, J., & Farley, D. (2010). Continuous delivery: Reliable software releases through build, test, and deployment automation. Addison-Wesley Professional.

Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The DevOps handbook: How to create world-class agility, reliability, and security in technology organizations. IT Revolution Press.

Knoche, H., & Hasselbring, W. (2018). Using microservices for legacy software modernization. IEEE Software, 35(3), 44-49. https://doi.org/10.1109/MS.2018.2141035

Kratzke, N., & Quint, P. C. (2017). Understanding cloud-native applications after 10 years of cloud computing: A systematic mapping study. Journal of Systems and Software, 126, 1-16. https://doi.org/10.1016/j.jss.2017.01.001

Mell, P., & Grance, T. (2011). The NIST definition of cloud computing (NIST Special Publication 800-145). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-145

Newman, S. (2015). Building microservices: Designing fine-grained systems. O'Reilly Media.

PwC. (2019). Financial services technology 2020 and beyond: Embracing disruption. PwC Global. https://www.pwc.com/gx/en/financial-services/assets/pdf/technology-2020-and-beyond.pdf

Richardson, C. (2018). Microservices patterns: With examples in Java. Manning Publications.

Sill, A. (2016). The design and architecture of microservices. IEEE Cloud Computing, 3(5), 76-80. https://doi.org/10.1109/MCC.2016.111

Trustgrid. (2020). Apiture case study: Zero trust networking for financial services. Trustgrid. https://trustgrid.io/case-studies/apiture/

VMware. (2020). Modernizing financial services: A path to cloud-native. VMware Tanzu. https://tanzu.vmware.com/content/white-papers/modernizing-financial-services

Zimmermann, O. (2017). Microservices tenets: Agile approach to service development and deployment. Computer Science - Research and Development, 32(3), 301-310. https://doi.org/10.1007/s00450-016-0337-0.