Universal Library of Engineering Technology Research Article

ISSN: 3064-996X | Volume 2, Issue 3

Open Access | PP: 94-99

DOI: https://doi.org/10.70315/uloap.ulete.2025.0203017

Scaling FinTech Startups from Zero to Millions of Users Engineering Challenges and Solutions

Andrii Humeniuk

Master Degree in Software Engineering, Lead Software Engineer, DASTA Incorporated ("dub"), New York, USA.
ORCID: https://orcid.org/0009-0002-0985-1146

Abstract

This article systematizes the engineering challenges encountered by the author when scaling fintech systems from inception to millions of users, including ensuring ultra-low latency, transactional consistency, and high availability. As a solution, a comprehensive framework is proposed, based on orchestrating containerized applications using Kubernetes, applying high-performance data handling patterns such as Command Query Responsibility Segregation (CQRS), and analyzing the architectural implications of integrating artificial intelligence (AI) technologies. The aim of the study is to systematize engineering challenges related to scaling FinTech platforms and to propose a framework of architectural patterns and cloud-oriented solutions to overcome them. The scientific novelty of the work lies in synthesizing fundamental distributed systems theory, modern cloud-native development practices, and forward-looking analysis of AI's architectural impact. This synthesis forms a comprehensive guide bridging the gap between academic theory and applied engineering practice, offering a strategic approach to building next-generation FinTech systems. The methodological basis of this work is founded on a synthetic approach that combines several research methods to form an analysis of the problem of scaling FinTech systems. The work demonstrates the practical significance of the proposed model, derived from the author's applied experience, for building fault-tolerant and high-performance financial systems that withstand extreme loads and comply with strict industry requirements.

Keywords: *Fintech, Scalability, High-Load Systems, Microservice Architecture, Kubernetes, CQRS, Low Latency, High Availability, Artificial Intelligence (AI), Risk Management.*

INTRODUCTION

Financial technologies (FinTech) are transforming the traditional financial industry by creating new models of customer interaction and opening access to financial services for broad user groups. Barroso M., Laborda J. , as a result of a systematic review, identified 188 concepts grouped into nine thematic clusters — issues, regulation, cooperation, and others — highlighting the interdisciplinarity and rapid development of research in the field of FinTech [1]. Khan R. A. et al. During the cartographic study, the existing software development practices in startups were analyzed, noting that only a small part demonstrate high scientific rigor, while the majority adapt to limited resources and dynamic market conditions [7]. At the same time, an analysis of 86 startup cases showed that the most severe form of technical debt accumulates in the testing subsystem, significantly hindering rapid product deployment to the market [5].

A central problem impeding scaling is the architectural legacy of many startups that begin with a monolithic architecture. This article builds on the author's prior research in cloud IT infrastructures [11] and hybrid financial storage architectures, extending those foundations into a comprehensive study of end-to-end scaling challenges in fintech platforms. While a monolith is effective in the early stages for rapid prototype development, as load and functional complexity grow, it becomes a technological barrier. Key bottlenecks include database contention, where competing operations slow down the entire system; tight coupling of components, in which even minor changes require a complete redeployment of the application, slowing down development cycles; and the inability to scale individual services independently. If the load increases on a single functional module, for example, on the payment processing service, in a monolithic system it becomes necessary to scale the entire application, leading to inefficient resource utilization.

The integration of AI/ML for tasks such as Risk Management, algorithmic trading, fraud detection, and market sentiment analysis creates new, computationally intensive, and

Citation: Andrii Humeniuk, "Scaling FinTech Startups from Zero to Millions of Users Engineering Challenges and Solutions", Universal Library of Engineering Technology, 2025; 2(3): 94-99. DOI: https://doi.org/10.70315/uloap.ulete.2025.0203017.

resource-demanding workloads. Machine learning models, particularly in the areas of Reinforcement Learning and Natural Language Processing, require access to large volumes of real-time data, placing extreme demands on throughput and latency in data processing systems. Legacy systems, designed without consideration for such workloads, are unable to handle them efficiently.

The aim of the study is to systematize engineering challenges related to scaling FinTech platforms and to propose a framework of architectural patterns and cloud-oriented solutions to overcome them.

The research objectives include: 1) analyzing trade-offs in transitioning from monolithic to microservice architecture; 2) substantiating the use of Kubernetes as a platform for orchestration and ensuring high availability; 3) detailed examination of high-performance data handling patterns such as sharding and CQRS; 4) analyzing the impact of AI/ML on architectural requirements; 5) validating the proposed framework based on a practical case of scaling a trading platform.

The scientific novelty of the work lies in synthesizing fundamental distributed systems theory, modern cloud-native development practices, and forward-looking analysis of AI's architectural impact. This synthesis forms a comprehensive guide bridging the gap between academic theory and applied engineering practice, offering a strategic approach to building next-generation FinTech systems.

The author's hypothesis is that sustainable scaling of FinTech platforms from zero to millions of users is possible only through the synergy of transitioning from a monolithic to a microservices architecture, orchestration via Kubernetes, the application of data processing patterns (CQRS, sharding, caching), and architectural solutions that take into account the real-time and high-availability requirements of AI/ML.

MATERIALS AND METHODS

The methodological basis of this work is founded on a synthetic approach that combines several research methods to form an analysis of the problem of scaling FinTech systems. The core of the study is a systematic literature review covering peer-reviewed scientific articles from leading academic databases. This method makes it possible to establish the theoretical foundation of the work, relying on well-established concepts and the latest research in the field of distributed systems and software engineering. In addition, content analysis is applied to authoritative technical documents, industry reports, and engineering blogs of leading technology companies. This approach ensures the enrichment of the theoretical basis with practical data and real-world examples of the implementation of the discussed technologies.

The source base for conducting the study is classified into three main categories, which ensures the completeness and reliability of the analysis:

Theoretical foundations: Fundamental works on the design of distributed systems, such as Designing Data-Intensive Applications by M. Kleppmann, as well as seminal scientific articles describing the architecture of such systems as Google Spanner and Amazon DynamoDB. These sources form the basis for understanding the key trade-offs in distributed systems, in particular between consistency, availability, and partition tolerance, known as the CAP theorem.

Market and industry analysis: Reports and publications from leading financial and consulting organizations, including the CFA Institute, World Economic Forum, JPMorgan, and McKinsey. Data from these sources are used for quantitative assessment of market trends, justification of the relevance of the topic, and analysis of the impact of AI on the financial sector.

Practical implementation patterns: Technical documentation from leading cloud providers (AWS, GCP) and publications from engineering blogs of technology leaders such as Uber Engineering. These materials provide evidence of the real-world application of the discussed architectural patterns and technologies, illustrating their advantages and disadvantages under production conditions.

Thus, the study is based on a multi-level source base, where academic works lay the theoretical foundation, industry reports provide context and statistical data, and technical documentation and industry case studies serve to demonstrate the practical applicability and validate the proposed solutions.

RESULTS AND DISCUSSION

The transition from a monolithic architecture, where the application is a single, indivisible block, to a microservice architecture consisting of a set of small, independently deployable services is a key strategic decision for scalable FinTech projects. However, this transition is not a panacea and is associated with a number of trade-offs that must be carefully analyzed.

The primary driver for migration is the increase in development velocity and the reduction of time-to-market. In a microservice architecture, teams can work on individual services autonomously, enabling them to develop, test, and deploy functionality independently. This eliminates the bottlenecks typical of monolithic systems, where a change in one module requires rebuilding and full regression testing of the entire application. However, this flexibility comes at the cost of significantly increased operational complexity. Managing dozens or hundreds of services, their network interactions, monitoring, and deployment requires a mature DevOps culture and a well-developed tooling platform.

This trade-off can be analyzed through the lens of the Team Cognitive Load concept described in the Team Topologies methodology. Cognitive load is divided into three types: intrinsic (the complexity of the business domain itself), extraneous (the complexity of tools and processes, such as deployment), and germane (related to learning and solving

new problems). The goal of effective organization is to minimize extraneous load so that engineers can focus on intrinsic and germane aspects. A poorly planned transition to microservices can catastrophically increase extraneous cognitive load: developers must think not only about business logic but also about network failures, service discovery, distributed transactions, and the challenges of debugging in a distributed environment.

Thus, the effectiveness of microservice adoption follows a nonlinear relationship. At the initial stage, decomposing a monolith increases team productivity by reducing code coupling and accelerating deployment cycles. However, as the number of services grows without corresponding advancements in platform engineering and automation, coordination and operational costs begin to rise exponentially. This leads to a situation where the benefits of team independence are offset by the complexity of managing the entire ecosystem, and the overall development speed may even decrease compared to a well-structured monolith. Success lies not in the mere fact of using microservices, but in creating a platform that abstracts and automates operational complexity, enabling teams to maintain their focus on business objectives [1, 3].

To minimize the risks associated with large-scale refactoring, the Strangler Fig Pattern is applied. This approach involves a gradual, incremental migration, in which new functionality is implemented as microservices operating in parallel with the legacy monolith. A special proxy layer (facade) intercepts incoming requests and routes them either to the old system or to the new service. Over time, more and more functionality "strangles" the monolith until it is completely decommissioned. This method enables continuous delivery of value to users while avoiding the risky "big bang" rewrite of the system (fig.1.).

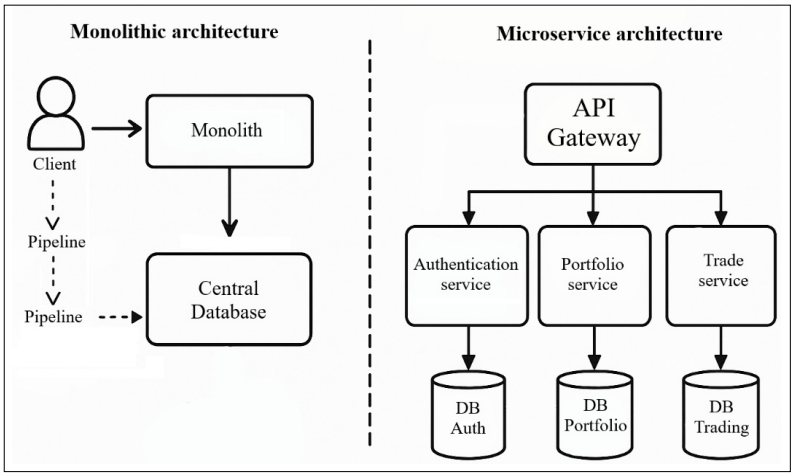

Fig.1. Architectural evolution from monolith to microservices in FinTech [1, 3, 7]

For the purpose of systematizing the identified problems and correlating them with the proposed engineering solutions, Table 1 presents an overview of the key challenges arising during the scaling of FinTech platforms, indicating their essence, approaches to mitigation, and the technologies used.

Table 1. Engineering challenges and solutions in scaling FinTech systems [1, 8, 10, 11]

Engineering challenge	Description of the problem	Proposed solution	Applied technologies/patterns
Limitations of monolithic architecture	High coupling of components, inability to scale individual modules	Migration to microservices with phased transition (Strangler Fig Pattern)	Spring Boot, gRPC/REST, API Gateway
Growth of operational complexity	Need to manage dozens of services, monitoring, and deployment	Container orchestration via Kubernetes, CI/CD, and IaC	Kubernetes (EKS), Terraform, GitLab CI/CD
Peak loads and volatility	Sharp traffic spikes during trading hours	Auto-scaling of pods and nodes	Kubernetes HPA, Cluster Autoscaler, AWS ALB
Databases as a bottleneck	Increased latency with the growth in the number of requests	Horizontal scaling and caching	Sharding (range/hash), Redis
CAP theorem conflict	Simultaneous requirement for strict consistency and high availability	CQRS with separation of commands and queries	Relational DB (CP) + NoSQL/ Cache (AP)
AI/ML integration	High load, low latency requirements	HTAP, streaming architecture	Apache Kafka, Flink, Kappa Architecture
Reliability and fault tolerance	Need to prevent cascading failures	Fault-tolerance and observability patterns	Circuit Breaker, Retry, Bulkhead, Prometheus, Jaeger

Kubernetes today acts as the de facto industry standard for orchestrating containerized applications and serves as a foundational platform that removes a significant portion of the complexities inherent to a microservice architecture. For FinTech systems, its properties — ensuring high availability and dynamic scalability — are critically important.

High availability is achieved through built-in self-healing mechanisms. In the event of a container (Pod) failure, the platform automatically restarts it; when a node (virtual machine) fails, the Pods running on it are rescheduled to healthy cluster nodes. To meet SLA and maximize fault tolerance, a typical practice is to deploy a Kubernetes cluster (for example, Amazon EKS) across multiple Availability Zones (AZ). Such a design guarantees that the failure of an entire data center will not lead to service downtime [3, 10].

Financial markets are characterized by high volatility, which gives rise to sharp and unpredictable load spikes, particularly during peak trading hours. Kubernetes addresses this problem through automatic scaling.

Horizontal Pod Autoscaler (HPA) regulates the number of service replicas (Pods) in accordance with the current load (for example, by CPU utilization). When the resources of existing nodes are no longer sufficient, Cluster Autoscaler automatically adds new nodes to the cluster. In combination, this provides elastic adaptation to real traffic, stable performance, and optimization of infrastructure costs. For correct routing of traffic to a dynamically changing number of Pods, an Application Load Balancer (ALB) integrated with an EKS cluster via the AWS Load Balancer Controller is used, which is considered a recommended practice.

Integration of Kubernetes with continuous integration and delivery (CI/CD) pipelines makes it possible to automate service version rollouts, rendering the process fast and predictable. At the same time, the infrastructure itself (clusters, networks, databases) is managed declaratively using Infrastructure as Code (IaC) tools such as Terraform. This ensures version control, reproducibility, and environment consistency, minimizing risks attributable to the human factor.

As the user base grows to millions of active clients, the database often becomes the bottleneck of a monolithic system. In distributed systems, strategies of horizontal scaling are applied to overcome this limitation [1, 9].

Database sharding implies the decomposition of a single logical database into many smaller and faster fragments (shards) distributed across different servers. This approach makes it possible to handle a substantially larger volume of data and number of requests than is possible on a single server. The key decision is the choice of the sharding key, the attribute by which data are distributed among shards. An incorrect choice leads to hot spots, when the main load is concentrated on one shard and the benefits of scaling are neutralized. Various strategies are used, for example, range-based or hash sharding; the choice is determined by the nature of queries to the data.

To achieve ultra-low latency in serving frequently requested data, caching in main memory is employed. Redis, being a high-performance in-memory data store, is ideally suited to this task. In fintech applications, Redis is used to cache user portfolio data, current market quotes, user profiles, and other entities requiring instantaneous access. This markedly reduces the load on the primary database and makes it possible to keep response time at the level of tens of milliseconds, which is critical for the user experience.

The Command Query Responsibility Segregation (CQRS) pattern provides a foundational architectural solution for systems with asymmetric read and write loads that are characteristic of the fintech domain. Its essence is the separation of the data model into two sides: the writing side (Command) and the reading side (Query). The write side processes operations that change the state of the system (create, update, delete) and is optimized for transactional integrity and consistency (ACID). The read side, by contrast, is aimed exclusively at executing queries and therefore can be denormalized, replicated, and materialized in forms convenient for fast data presentation without complex joins.

This separation entails an important trade-off — eventual consistency. Data from the write side are propagated asynchronously to the read side, as a rule through a message broker, and therefore for a short period the read model may contain slightly stale information. For most user scenarios in fintech (for example, viewing the transaction history or the current state of a portfolio) a delay of several hundred milliseconds is acceptable, whereas the gains in performance and scalability are substantial [6, 8].

The approach simultaneously provides a practical resolution of the dilemma formulated by the CAP theorem. Financial systems require properties that are difficult to combine: strong consistency for transactional operations and high availability with low latency for reading user data. CQRS makes it possible to satisfy these requirements by applying different consistency models to different parts of the system: the command side is built as a CP system (Consistency/Partition Tolerance), for example on a relational DBMS that ensures the integrity of each transaction, whereas the read side is implemented as an AP system (Availability/Partition Tolerance) on a replicated NoSQL store or cache, providing fast and fault-tolerant access to data for presentation. Thus, CQRS does not circumvent the CAP theorem but offers a pragmatic method for designing a system under conflicting business requirements [2, 7].

It should also be noted that the adoption of artificial intelligence has ceased to be optional and has become a key factor of competitiveness in the fintech sector. At the same time, AI and machine learning models impose qualitatively new requirements on the system architecture, as reflected in Fig. 2.

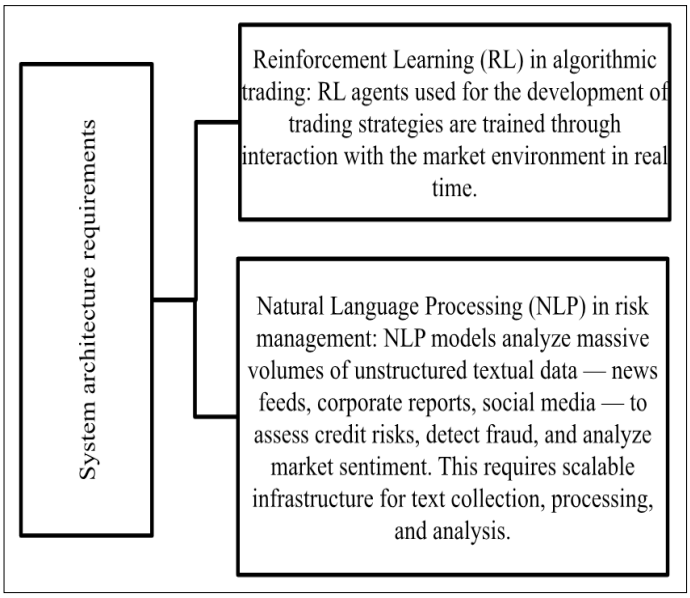

Fig.2. Requirements for the AI model [2, 7, 8]

To ensure resilience and observability, the system must be designed with fault tolerance prioritized as a fundamental property of the architecture. Within this logic, a set of proven patterns is applied. Circuit Breaker prevents cascading failures: when a series of errors occurs on a call to an external service, the breaker moves to the open state, and subsequent requests are rejected immediately without imposing load on a potentially faulty component; after an interval it transitions to the half-open state for a trial request and, if that succeeds, closes again. Retry addresses transient failures (for example, network-related): the client repeats the failed request a limited number of times, typically with an exponential pause between attempts so as not to overload the service during the recovery phase. Bulkhead provides resource isolation — separate thread or connection pools — by service, preventing a situation in which a local failure or a load spike exhausts system-wide resources, analogous to how watertight compartments on a ship localize a breach.

In distributed landscapes with dozens of microservices, a complete enumeration of all failure scenarios is fundamentally unattainable; therefore, modern practice relies on observability as a property that enables inference of internal state and root causes of incidents directly in production [3, 4]. For mature SRE teams in the financial sector, a key discipline is chaos engineering: the controlled introduction of faults into the live environment (for example, disabling a service or injecting network latency) to verify the effectiveness of fault tolerance mechanisms, monitoring, and alerting. This approach makes it possible to identify and eliminate architectural vulnerabilities before they materialize as real incidents.

CONCLUSION

Scaling a FinTech startup from concept to a platform with millions of users does not boil down to simply increasing computing resources; it constitutes a multidimensional engineering task that requires revisiting the initial architectural assumptions. The analysis conducted shows that sustainable growth is possible only when moving from a monolith to a distributed microservices architecture managed by Kubernetes-level cloud orchestrators. Such a platform provides the necessary elasticity, fault tolerance, and a high degree of automation that meet the industry's strict requirements for availability and performance.

The second key finding concerns specialized data patterns in high-load systems. Applying CQRS makes it possible, at the architectural level, to separate the contradiction between strict transactional consistency and the need for minimal read latency, rationally managing the trade-offs predetermined by the CAP theorem. In combination with sharding and caching, this forms a reliable foundation for a high-performance data layer.

Finally, it is shown that artificial intelligence is no longer an auxiliary function and is becoming a driver that determines architectural requirements. The need for real-time data processing for AI/ML models forces a rethinking of the traditional separation of transactional and analytical contours, stimulating the adoption of HTAP approaches and streaming architectures.

The practical significance of the proposed framework lies in providing technical leaders and architects with a strategic roadmap for well-founded decision-making. It systematizes the trade-offs between development speed, operational complexity, and reliability, enabling the construction of an engineering strategy closely aligned with the company's business objectives.

The study validates a practical framework for scaling fintech platforms, unifying distributed systems theory with applied engineering practice. By systematizing trade-offs and integrating Kubernetes, CQRS, sharding, caching, and AI/ML, the author contributes a roadmap for technical leaders scaling platforms from zero to millions of users.

The following directions for further research are identified: a deeper study of the impact of serverless architectures on the operating models of FinTech companies; an analysis of the specific challenges of integrating large language models (LLM) into low-latency systems; and a study of the adaptation of architectural solutions to the dynamically changing regulatory environment for the use of AI in finance.

REFERENCES

1. Barroso, M., & Laborda, J. (2022). Digital transformation and the emergence of the fintech sector: Systematic literature review. Digital Business, 2(2), 100028.. https://doi.org/10.1016/j.digbus.2022.100028
2. Vijayagopal, P., Jain, B., & Ayinippully Viswanathan, S. (2024). Regulations and Fintech: A Comparative Study of the Developed and Developing Countries. Journal of Risk and Financial Management, 17(8), 324. https://doi.org/10.3390/jrfm17080324

3. Kumar, G. (2025). Architecting scalable and resilient fintech platforms with AI/ML integration. Journal of Innovative Science and Research Technology, 10(4), 3073-3084. https://doi.org/10.38124/ijisrt/25apr2359

4. Lange, F., et al. (2023). Demystifying massive and rapid business scaling—An explorative study on driving factors in digital start-ups. Technological Forecasting and Social Change, 196, 122841. https://doi.org/10.1016/j.techfore.2023.122841

5. Abrahamsson, V., & Holmqvist, V. (2023). Technical debt in Swedish tech startups: Uncovering its emergence, and management processes, 34-60 .

6. Wetzel, T., & Eiche, J. (2024). Challenges of start-ups—An analysis of individually tailored recommendations based on the development phases, branches, business models and founding teams. Open Journal of Business and Management, 12(3), 1556–1585. https://doi.org/10.4236/ojbm.2024.123084

7. Khan, R. A., et al. (2021). Systematic mapping study on security approaches in secure software engineering. IEEE Access, 9, 19139–19160. https://doi.org/10.1109/ACCESS.2021.3052311

8. Tamraparani, V. (2024). AI and Gen AI application for enterprise modernization from complex monolithic to distributed computing in fintech and healthtech organizations. Journal of Artificial Intelligence, Machine Learning and Data Science, 2(2), 1611-1617.

9. Bolgov, S. (2024). Creating infrastructure for scalable fintech solutions: Technical and organizational aspects. ISJ Theoretical & Applied Science, 12(140), 354–358

10. Alt, R., Fridgen, G., & Chang, Y. (2024). The future of fintech—Towards ubiquitous financial services. Electronic Markets, 34,3.

11. Telenik S.T. et al. Development and research of models, methods and technologies of planning, programming and management cloudy IT-infrastructures. Retivered from:https://nrat.ukrintei.ua/en/searchdoc/0216U005227/ (date of access: 10.07.2025)