240. MICROSERVICE ARCHITECTURE IN USER APPLICATIONS

Evteev V.K.
Belarusian State University of Informatics and Radioelectronics
Minsk, Republic of Belarus
Sinkevich L.E. – Senior Lecturer

The problem of choosing the right type of architecture while creating an application is discussed in this paper. The history of using microservices architecture for application development, its advantages and disadvantages in comparison with monolithic one are shown.

Modern user applications must cater to millions of concurrent users while remaining highly available and easy to update. Traditional monolithic architectures often struggle to meet these demands – a minor change in one module can require redeploying the entire application, creating bottlenecks in both development and scalability. This paper provides a comprehensive look at microservice architecture in user applications, exploring how it addresses the limitations of monolithic systems while also examining the trade-offs involved, such as increased operational overhead and the complexities of managing distributed data.

The term "microservices" was first introduced at a software workshop held in 2011. Several important concepts played a role in the emergence of microservices, particularly domain-driven design – model-based development and the principles of bounded contexts and continuous integration. “Methods of designing for failure, data isolation, infrastructure automation, large-scale agile development, cross-functional team building, and responsibility for the entire product lifecycle have also had a significant impact” [1].

The real-world impact of microservice architecture is best seen in the companies that have successfully adopted it. Netflix is a classic example – It moved from a monolithic data center model to a cloud-native microservice architecture that now serves hundreds of millions of subscribers worldwide. Amazon took a similarly ambitious path, restructuring its retail platform into hundreds of loosely coupled services. This allowed individual teams own specific business capabilities – like order management or inventory tracking – and innovate at massive scale. In fintech, Hastee, a UK-based platform for instant wage access, built its application on microservices from day one, which helped it gain early user traction, scale with complex payroll needs, and keep evolving its user experience smoothly. “South Korea's Lotte Group shows the power of this approach in e-commerce: its platform now handles over a billion API calls daily while keeping response times under 1.2 seconds for all user queries” [2].

So how does microservice architecture stack up against the traditional monolithic approach? There are some clear advantages. That is a huge contrast with monoliths, where any surge means scaling the whole application, wasting resources and driving up costs. “Microservices can accelerate innovation by letting teams develop, test, and deploy independently. Each team can select the right tool for the job – Python for data work, Node.js for APIs, Go for performance-critical tasks – without being locked into one stack” [3]. Fault isolation is another significant advantage: if one service fails, the rest keep running. A broken recommendation engine will not take down your checkout flow, whereas in a monolith, one bug can crash everything. Additionally, smaller codebases are just easier to work with than sprawling monolithic ones.

Nevertheless, these benefits are not without associated costs. Operational complexity increases significantly in a distributed world. Such systems require service discovery, distributed tracing, container orchestration (like Kubernetes), and solid monitoring, all of which require specialized skills that many teams do not have. “Data consistency gets trickier – ACID transactions that worked seamlessly in a monolith now need eventual consistency models and patterns like Saga to keep data straight across services” [4].

To sum up, microservice architecture is a powerful evolution in how we build software, and companies like Netflix, Amazon, and Lotte show just transformative it can be when applied to large, complex domains with clear service boundaries and mature teams. But the choice between microservices and a monolith should not be driven by hype – this should be determined based on organization's actual requirements, team size, and business goals. The move to microservices makes sense when real signals emerge – team growth beyond eight to ten people, deployment coordination becoming a bottleneck, or components requiring divergent scaling approaches – a more distributed architecture may be warranted. Sound architecture is defined not by technical elegance, but by business velocity. The objective is to build systems that enable rapid delivery, reliability, and adaptability. By understanding both advantages and drawbacks, engineering leaders can make decisions aligned with the organization's capabilities and long-term objectives.

References:

1. Microservices: The Path Taken and Future Goals. – URL: https://www.osp.ru/os/2018/03/13054404#1 (date of access: 12.03.2026).

2. Lotte scales commerce traffic beyond 1B calls/day with Tyk. – URL: https://tyk.io/case-studies/lotte/?utm_campaign=comparison-and-pricing (date of access: 13.03.2026).

3. Microservices vs. Monoliths: What’s Best for Scalable Apps? – URL: https://catalyst.zoho.com/blog/monolith-vs-microservices.html (date of access: 13.03.2026).

4. Monolithic vs microservices architecture: when to choose each approach. – URL: https://getdx.com/blog/monolithic-vs-microservices/#content (date of access: 13.03.2026).