FROM MICROSERVICES TO SERVERLESS: A TEN-YEAR RETROSPECTIVE AND FUTURE ROADMAP (2016-2026): A CASE STUDY OF BACKEND SERVICE ARCHITECTURE IN TELECOM CUSTOMER SUPPORT PLATFORMS

Dorëart REÇICA

Department of Informatics, State University of Tetovo
recicadoreart@gmail.com

Abstract

Over the past decade, enterprise software architecture has evolved from traditional monolithic systems and Service-Oriented Architecture (SOA) toward microservices and serverless computing. Service-oriented approaches established foundations for distributed and interoperable systems, while microservices introduced greater scalability, flexibility, and independent deployment [1], [4]. This evolution has also created challenges related to service communication, orchestration, data management, reliability, and system complexity [5], [7]. In telecommunications, customer support platforms depend on interconnected backend services for authentication, customer relationship management, network diagnostics, ticket management, notifications, and knowledge retrieval.

This paper presents a ten-year retrospective of the evolution from microservices to serverless computing between 2016 and 2026. A qualitative case study of a representative telecom customer support platform is used to examine how distributed backend services cooperate to process customer requests. The study evaluates scalability, maintainability, deployment flexibility, operational complexity, data management, and cost efficiency. Particular attention is given to service orchestration, microservice data management, and Function-as-a-Service (FaaS) [9]. The findings indicate that microservices remain suitable for long-running and business-critical services, while serverless computing can complement them for short-duration and event-driven workloads. The research concludes that hybrid architectures combining microservices, containers, serverless functions, and reliable service orchestration provide a promising direction for scalable telecom customer support platforms [8], [11].

Keywords: Microservices, Serverless Computing, Cloud Computing, Function-as-a-Service (FaaS), Service-Oriented Architecture, Backend Services, Telecom Customer Support, Cloud-Native Applications, Distributed Systems, Enterprise Software Architecture

1. Introduction

1.1 Background of the Study: The development of enterprise software architecture has changed considerably over the last decade. Traditional monolithic applications initially provided a straightforward approach in which the main business logic, data access, and application functionality were implemented and deployed as a single system. As enterprise applications became larger and more complex, however, organizations increasingly adopted Service-Oriented Architecture (SOA) and other distributed approaches to improve modularity, interoperability, and service reuse [1], [2]. Research on service orchestration and choreography also demonstrated the importance of effectively coordinating independent services when building distributed applications [3].

With the rise of microservices, a new architectural style has been introduced to handle large-scale cloud applications. As described by Lewis and Fowler in [5], microservices are organized in a manner of a collection of small applications, each consisting of independently deployable services. Each service typically corresponds to a particular business capability and is implemented, deployed, scaled and managed as if it were a standalone application. Due to the differing performance, availability and scaling requirements of various parts of a system, such an architectural style has recently gained a lot of popularity [6].

The adoption of microservices has nevertheless introduced new technical challenges. Distributing application functionality across many services increases the complexity of

communication, monitoring, testing, deployment, and data management. Each service may also require its own data storage strategy, making data consistency and persistence important architectural considerations [7]. Recent research has therefore examined not only the benefits of microservices but also their architectural evolution and the challenges involved in maintaining distributed systems over time [8].

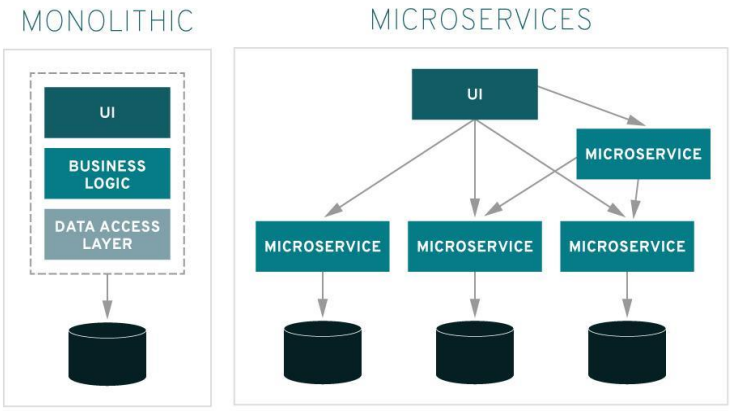


Figure 1. Evolution of Enterprise Backend Architecture from Monolithic Systems to Serverless Computing

Serverless computing has recently gained significant traction in the area of cloud-native applications. Serverless platforms, often referred to as Function-as-a-Service (FaaS) platforms, allow application functionality to be deployed as managed functions written in a variety of languages that are executed in the cloud without the need to provision and manage underlying servers [10]. Research has also identified serverless computing as a complementary technology to microservices rather than necessarily a complete replacement for them [11].

1.2 Problem Statement: Telecommunications customer support platforms must process large volumes of customer interactions while providing fast, reliable, and consistent services. A single customer request may require several backend systems to communicate with one another, including authentication, CRM, network diagnostics, ticketing, knowledge management, and notification services. As the number of services increases, maintaining reliable communication and consistent performance becomes increasingly difficult [8].
While microservices architectures can address several limitations of monolithic architectures by decomposing an application into a set of independent services, each deployable in isolation, the resulting architecture introduces several new challenges for the operational management of the application, including service discovery, distributed monitoring, failures, data consistency, and inter-service communication. Serverless computing can be an alternative architecture style for specific use cases by shifting the focus from managing a pool of servers to managing the infrastructure required to support an application, and then automatically scaling to handle changes in demand.
The main problem addressed by this research is therefore determining how microservices and serverless computing can be effectively combined within telecom customer support backend platforms, while maintaining scalability, reliability, maintainability, and efficient service coordination.

1.3 Aim and Objectives: The main aim of this research is to examine the evolution of backend software architecture from microservices to serverless computing during the period 2016–2026 and to evaluate how these architectural approaches can be applied within telecom customer support platforms. The research focuses on the relationship between architectural design,

scalability, maintainability, service coordination, and operational efficiency in distributed telecom environments.
The specific objectives of this research are:

- To analyze the evolution from monolithic and Service-Oriented Architecture toward microservices and serverless computing [1].
- To examine the main architectural characteristics and principles of microservices.
- To investigate the role of containers and orchestration technologies in modern microservice environments [8], [10].
- To examine the principles, benefits, and limitations of serverless and Function-as-a-Service architectures [9], [11].
- To compare microservices and serverless computing in terms of scalability, performance, maintainability, deployment, complexity, and cost.
- To analyze how distributed backend services can support telecom customer support operations.
- To investigate the role of service orchestration and choreography in coordinating distributed telecom services .
- To identify suitable use cases for combining microservices and serverless functions within a telecom customer support platform.
- To identify the main challenges associated with adopting hybrid microservice-serverless architectures.

1.4 Research Question: The main research question addressed by this study is:
How has backend service architecture evolved from microservices toward serverless computing between 2016 and 2026, and how can these approaches improve the scalability and operational efficiency of telecom customer support platforms?
The following sub-questions support the main research question:

- How did microservices address limitations associated with traditional monolithic and SOA-based architectures [1]?
- What are the main advantages and limitations of microservice architectures in distributed telecom systems?
- How does serverless computing differ from traditional microservice deployment models?
- Which telecom customer support workloads are most suitable for microservices and which are suitable for serverless functions?
- How do service orchestration and choreography influence communication between distributed backend services [3], [4]?
- How can microservices and serverless functions be combined within a single telecom customer support platform?
- What challenges related to scalability, reliability, data management, security, monitoring, and operational complexity must be considered?

2. Literature Review

This chapter reviews the evolution of distributed software architecture from Service-Oriented Architecture (SOA) to microservices and serverless computing. It focuses on the main characteristics, benefits, challenges, and applications of these approaches in modern cloud-based and telecom systems [5].

2.1 Service-Oriented Architecture: Service-Oriented Architecture (SOA) structures applications as a collection of independent and reusable services that communicate through defined interfaces. SOA improved interoperability, service reuse, and integration between distributed applications.
Service orchestration and choreography are important aspects of SOA because multiple services often need to cooperate to complete a business process. Research by Dika and Halili examined service orchestration, choreography, and execution planning in distributed web services [4].

2.2 Microservice Architecture: Microservices divide an application into smaller, independently deployable services, with each service generally responsible for a specific business capability [5], [6]. This approach allows individual services to be developed, deployed, and scaled independently.
Key benefits include:

- Independent deployment
- Improved scalability
- Fault isolation
- Technology flexibility
- Easier maintenance

However, microservices also introduce challenges such as distributed communication, monitoring, security, testing, and data consistency [6], [7].
In telecom customer support, microservices can separate functions such as authentication, CRM, network diagnostics, ticket management, and notifications.

2.3 Serverless Computing: Serverless computing enables applications to be run without the developer having to manage servers. It consists of so-called Functions-as-a-Service (FaaS). These FaaS functions are executed by serverless systems in the form of so-called events or API calls [9], [10].
Important advantages include:

- Automatic scaling
- Reduced infrastructure management
- Event-driven execution
- Flexible resource utilization
- Pay-per-use pricing

However, serverless systems can introduce challenges involving performance, monitoring, security, vendor dependency, and function coordination.

2.4 Microservices and Serverless: Microservices and serverless computing can be used together rather than being treated as competing approaches. Microservices are well suited to long-running business functions, while serverless functions can handle short-lived and event-driven workloads [13].
For example, a telecom customer support platform could use microservices for CRM, authentication, and network diagnostics, while serverless functions could process notifications, scheduled tasks, or event-triggered operations.

Table 1. Comparison of Microservices and Serverless Computing

Characteristic	**Microservices**	**Serverless**
Deployment	Independent services	Individual functions
Scaling	Service-based	Automatic

Characteristic	Microservices	Serverless
Infrastructure	Managed by organization	Managed by provider
Workload	Long-running services	Event-driven tasks
Cost model	Provisioned resources	Pay-per-use
Control	Higher	Lower

2.5 Service Orchestration in Distributed Systems: Service orchestration and choreography remain important in modern distributed architectures because multiple independent services must coordinate to complete customer requests. Earlier research by Dika and Halili on web-service orchestration and choreography provides a foundation for understanding these coordination challenges [3].
In a telecom support platform, a single customer request may involve authentication, CRM lookup, network diagnostics, ticket creation, and notification services.

3. Methodology

This chapter describes the methodology used to examine the evolution of microservices and serverless computing and their application in telecom customer support platforms. The research uses a qualitative case study supported by a review of existing academic literature on distributed systems, SOA, microservices, and serverless computing [9].

3.1 Research Approach: A qualitative case study approach was selected because it allows the research to examine how different backend services interact within a realistic telecom customer support environment.

The study focuses on a representative platform containing several backend components:

1. **Authentication Service** - manages customer authentication and access.
2. **CRM Service** - provides customer and service information.
3. **Network Diagnostics Service** - analyzes technical connectivity issues.
4. **Ticket Management Service** - records and manages customer incidents.
5. **Knowledge Service** - provides troubleshooting and service information.
6. **Notification Service** - sends customer notifications using event-driven functions.

The architecture combines microservices for core business functions with serverless functions for suitable short-lived and event-driven operations [13].

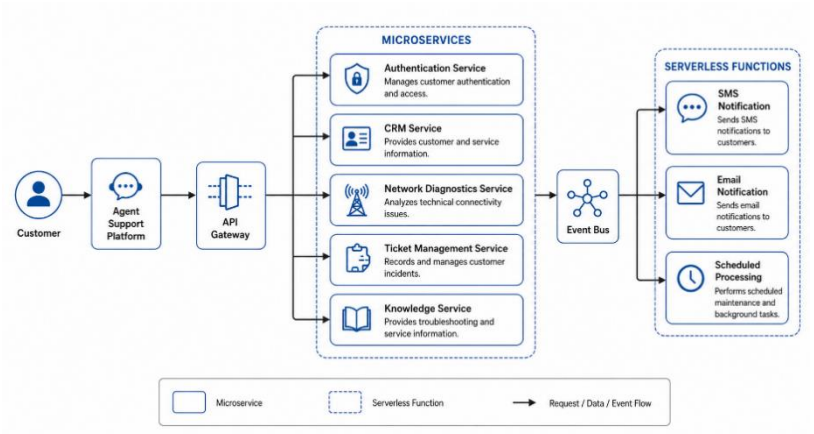


Figure 2. Proposed Hybrid Microservice-Serverless Telecom Architecture

3.2 Data Collection: Data for the study was collected from three main sources:

- Academic literature covering SOA, microservices, serverless computing, containers, and distributed systems [5].
- Observation of telecom support workflows to identify common backend services and interactions.
- Case study analysis of how different architectural approaches could support customer support operations.

The study uses a representative architecture rather than confidential internal company infrastructure.

3.3 Data Analysis: The collected information was analyzed according to the following characteristics:

1. **Scalability** - ability to handle changes in workload.
2. **Performance** - response and processing requirements.
3. **Maintainability** - ease of updating individual components.
4. **Deployment flexibility** - ability to deploy services independently.
5. **Operational complexity** - effort required to manage the architecture.
6. **Cost efficiency** - infrastructure and resource utilization.

These characteristics were selected because they are important when comparing microservices and serverless architectures.

3.4 Case Study Design: The case study follows a typical telecom support scenario in which a customer reports a technical problem.
The process consists of:
Customer Request → Authentication → CRM Lookup → Network Diagnostics → Resolution → Ticket Update → Customer Notification
Microservices are used for persistent and business-critical functions, while serverless functions are considered for event-driven operations such as notifications and scheduled processing.
The case study is used to compare how these architectural approaches can support the same customer support workflow while considering scalability, flexibility, reliability, and operational complexity.

3.5 Ethical Considerations: The research does not use personally identifiable customer information or confidential technical infrastructure.
The case study represents a generalized telecom environment based on common industry practices. Company-specific systems and implementation details are not disclosed.

4. Analysis and results

This chapter analyzes how microservices and serverless computing can be applied to backend services within a telecom customer support platform. The analysis focuses on scalability, performance, maintainability, deployment flexibility, and operational complexity. The results are based on the architecture described in the case study and the characteristics identified in the reviewed literature.

4.1 Microservices for Core Business Functions: Microservices are particularly suitable for the core functions of the telecom support platform because these functions require persistent state and continuous availability.

For example, the CRM service can be scaled independently when customer information requests increase, without requiring the network diagnostics or ticket management services to scale at the same rate.
The same principle can be applied to network diagnostics. During periods of increased technical incidents, additional instances of the diagnostic service can be deployed while other services remain unchanged.
This demonstrates one of the main advantages of microservices: independent scalability [6], [12].

4.2 Serverless Functions for Event-Driven Operations: Serverless computing is more appropriate for operations that are short-lived or triggered by specific events. In the proposed architecture, notifications are a suitable example.
When a support agent completes a customer case, an event can trigger a serverless function that sends an SMS or email to the customer. The function does not need to run continuously and can be executed only when required.

4.3 Comparison of Microservices and Serverless: The case study demonstrates that neither architecture is optimal for every backend function. Microservices provide greater control and are appropriate for persistent business services, while serverless is advantageous for event-driven and intermittent workloads [13].

Table 2. Architecture Selection for Telecom Backend Services

Service	**Architecture**	**Main Reason**
Authentication	Microservice	Continuous availability
CRM	Microservice	Persistent business data
Network Diagnostics	Microservice	Complex processing
Ticket Management	Microservice	Stateful workflow
Knowledge Service	Microservice	Continuous availability
SMS Notification	Serverless	Event-driven
Email Notification	Serverless	Event-driven
Scheduled Reports	Serverless	Periodic execution

The results indicate that a hybrid architecture provides greater flexibility than attempting to implement the entire platform using only microservices or only serverless functions.

4.4 Scalability and Performance: Scalability is one of the strongest advantages of both approaches, although they achieve it differently. Microservices allow specific services to be replicated according to demand, while serverless platforms automatically allocate resources when functions are invoked [6].
For example, if the number of customer technical incidents increases, additional instances of the Network Diagnostics service can be deployed. At the same time, a large number of notification requests can trigger additional serverless function executions without manually provisioning servers.
However, serverless functions may introduce startup latency and performance variability, particularly when functions have not been recently executed [10], [14]. Therefore, latency-sensitive and continuously active operations may be better suited to traditional microservices.

4.5 Service Communication and Failure Handling: The distributed nature of the proposed architecture means that communication between services becomes an important consideration. A customer request may require several services to respond before the support process can be completed.

If one service becomes unavailable, the architecture should prevent the failure from unnecessarily affecting unrelated services. Techniques such as retries, timeouts, circuit breakers, and asynchronous messaging can help improve system resilience [15].

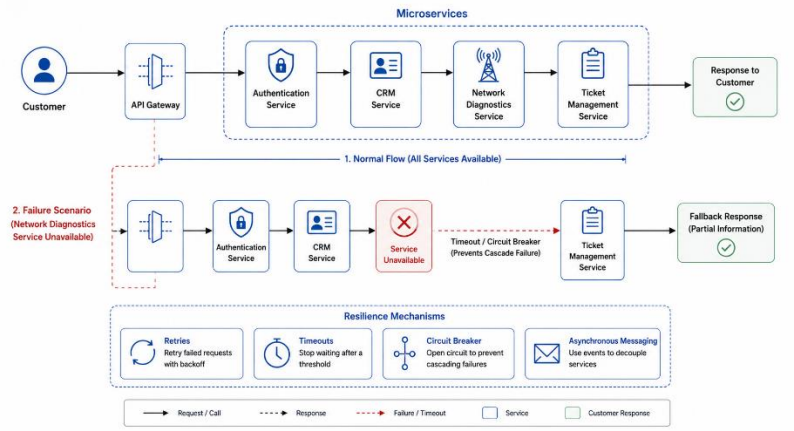


Figure 3. Telecom Customer Support Request and Failure Handling Flow

The analysis shows that failure isolation is particularly important in telecom systems because customer support operations should continue even when an individual backend component experiences an issue.

5. Discussion

This chapter discusses the results presented in Chapter IV and evaluates the practical implications of combining microservices and serverless computing within a telecom customer support platform.

5.1 Efficiency and Scalability: Microservices provide the ability to scale individual backend components independently. This is particularly useful in telecom support, where services such as network diagnostics may experience significantly different workloads from CRM or ticket management [12].

Serverless functions provide additional flexibility for event-driven tasks. Their automatic scaling can be useful when workloads are unpredictable, such as sending large numbers of customer notifications after a service incident [9], [10].

The main advantage of the hybrid approach is therefore the ability to select the most appropriate architecture for each workload.

5.2 Maintainability and Flexibility: Separating the backend into independent services can make development and maintenance easier because changes to one service do not necessarily require changes to the entire platform.

However, this benefit comes with increased operational complexity. Developers and administrators must manage service communication, monitoring, security, and distributed failures. Polyglot persistence can also increase flexibility while making data management more complex [7].

Therefore, architectural independence should be balanced against the additional complexity created by a larger number of components.

5.3 Reliability and Service Coordination: Reliability is particularly important in telecom customer support because backend failures can directly affect the ability of agents to assist customers.
Service orchestration, choreography, asynchronous messaging, and fault-isolation mechanisms can help coordinate distributed services and reduce the impact of individual failures [4], [15].
The case study indicates that critical services such as authentication, CRM, and network diagnostics should remain continuously available, while less critical event-driven processes can be separated into serverless functions.

5.4 Implications for Telecom Customer Support: The findings have several practical implications for telecom customer support platforms.
A hybrid architecture could allow core services such as CRM, authentication, and network diagnostics to operate as microservices while using serverless functions for notifications, scheduled processing, and other event-driven operations.

5.7 Challenges and Limitations: Despite the benefits, several challenges remain:

- **System complexity:** A larger number of distributed components increases monitoring and troubleshooting requirements.
- **Data consistency:** Independent services can make cross-service transactions more difficult.
- **Serverless limitations:** Cold starts, execution constraints, and provider dependency may affect certain workloads [14].
- **Security:** Each service and API introduces additional security boundaries.
- **Integration**: Combining microservices and serverless functions requires reliable communication and event management.

The case study is also limited because it represents a generalized telecom environment rather than a complete production architecture. The results therefore provide architectural insights rather than quantitative performance benchmarks.

6. Conclusion and future work

6.1 Conclusion: This research examined the evolution of backend service architecture from microservices toward serverless computing between 2016 and 2026, with a focus on telecom customer support platforms.
The study showed that microservices provide important benefits such as independent deployment, scalability, service isolation, and flexibility. However, they also introduce challenges related to distributed communication, data management, monitoring, and operational complexity [15].
Serverless computing further extends the abstraction of underlying infrastructure by automatically managing resources and scaling individual functions as needed. In the case study serverless computing was found to be especially suitable for event-driven processing such as notifications, scheduled processing and background processing.
In summary, the views of microservices and serverless computing can be very different, but the two concepts are not mutually exclusive. When choosing between them, microservices can be used to run the core functions of a telecoms company, such as CRM, authentication, diagnostics and ticket management, while running event-driven workloads in serverless functions where appropriate.

6.2 Future Work: Potential areas for further research include:

- Quantitative evaluation: Measure response time, scalability, resource consumption, and cost between microservice and serverless implementations.
- Cloud provider comparison: Compare serverless platforms from providers such as AWS, Microsoft Azure, and Google Cloud.
- Security analysis: Investigate security risks associated with APIs, distributed services, containers, and serverless functions.
- AI integration: Examine how AI-based monitoring and automated decision-making could improve the management of distributed telecom services.
- Real-world implementation: Develop and evaluate a production-like telecom support platform using the proposed hybrid architecture.

References

[1] M. P. Papazoglou and W.-J. van den Heuvel, "Service oriented architectures: Approaches, technologies and research issues," The VLDB Journal, vol. 16, pp. 389–415, 2007.
[2] P. Iseni and F. Halili, "Reliable Service-Oriented Architecture for NASA's Mars Exploration Rover Mission," in Proc. 11th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 2022, pp. 1–5, doi: 10.1109/MECO55406.2022.9797097.
[3] A. Dika and F. Halili, "Integrated orchestration of web services and the impact of the query optimization," in Proc. 8th International Conference on Computing Technology and Information Management, Seoul, South Korea, 2012, pp. 702–708.
[4] F. Halili and A. Dika, "Choreography of web services and the impact of estimation of execution plan," in Proc. International Conference on Information Technology and e-Services, Sousse, Tunisia, 2012, pp. 1–7, doi: 10.1109/ICITeS.2012.6216619.
[5] J. Lewis and M. Fowler, "Microservices: A definition of this new architectural term," MartinFowler.com, 2014.
[6] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," Journal of Systems and Software, vol. 150, pp. 77–97, 2019, doi: 10.1016/j.jss.2019.01.001.
[7] M. K. Halili, F. Halili, D. M. Veliu, and A. Nuhiji, "Polyglot persistence in microservices: Managing data diversity in distributed systems," in Proc. 2025 IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS), Bali, Indonesia, 2025, pp. 247–253.
[8] D. Bernstein, "Containers and cloud: From LXC to Docker to Kubernetes," IEEE Cloud Computing, vol. 1, no. 3, pp. 81–84, 2014.
[9] E. van Eyk et al., "The SPEC-RG reference architecture for FaaS: From microservices and containers to serverless platforms," IEEE Internet Computing, vol. 23, no. 6, pp. 7–18, 2019, doi: 10.1109/MIC.2019.2952061.
[10] M. Shafiei, A. Khonsari, and M. Mousavi, "Serverless computing: A survey of opportunities, challenges, and applications," ACM Computing Surveys, 2022.
[11] E. Jonas et al., "Cloud programming simplified: A Berkeley view on serverless computing," arXiv preprint arXiv:1902.03383, 2019.
[12] P. Di Francesco, I. Malavolta, and P. Lago, "Research on architecting microservices: Trends, focus, and potential for industrial adoption," in Proc. IEEE International Conference on Software Architecture (ICSA), 2017.
[13] E. van Eyk, L. Toader, S. Talluri, L. Versluis, A. Iosup, and S. Eismann, "Serverless is more: From PaaS to present cloud computing," IEEE Internet Computing, 2018.
[14] S. Hendrickson et al., "Serverless computation with OpenLambda," in Proc. 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud), 2016.
[15] N. Dragoni et al., "Microservices: Yesterday, today, and tomorrow," in Present and Ulterior Software Engineering, Springer, 2017, pp. 195–216.