WORLD COMET RESEARCH GROUP

International Journal of Applied Data Science & Modern Computing
Vol. 5, No. 2, 2022
Doi: 10.67228/30713498/IJADSMC-2022PII0L1Z
PP. 01-12

Original Article

Scalable Microservices Architecture for Data-Intensive Applications

Dr. Siti Aisyah Rahman

Department of Education, Nusantara State University, Kuala Lumpur, Malaysia.

Received: 06-06-2022 Revised: 06-28-2022 Accepted: 07-03-2022 Published: 07-05-2022

ABSTRACT

The modern software architectures have never had higher requirements than they do currently as the result of the rapid growth of data-intensive applications in the fields of cloud computing, Internet of Things (IoT), artificial intelligence, and large-scale web services. Amongst the problems that traditional monolithic systems cannot cope with are scalability, fault tolerance and on-going deployment demands using extensive data workloads. The consequence of this is that microservice architecture has become an architecture with potential to make systems that are scalable, resilient, and maintainable. The paper provides a detailed research of scalable microservices architecture specifically to the data intensive applications. The suggested architecture breaks the complex applications in loosely coupled, independently deployable services, which are specialized in the processing of specific data. Such fundamental architectural values as service granularity, data decentralization, containerization, orchestration, and asynchronous communication are discussed in detail. An overlay-based architectural design is presented to help scalability horizontally and provide high availability as well as efficient data management. Also, the analysis of current architectural methods is done in this study based on the comprehensive literature review that finds out the limitations in scalability, consistency, and operational complexity. An end-to-end process is suggested, where distributed data stores are combined with event-driven communication and automatic scaling processes. The analysis of performance has shown that the proposed solution is much better in terms of throughput, latency and fault isolation than the conventional architectures. These findings prove that microservices architecture with proper design presents a solid base of scalable data-intensive systems. The paper is useful to the researchers and practitioners who need to develop high-performance, cloud-native applications using micro services.

KEYWORDS

Microservices, Scalability, Data-Intensive Systems, Distributed Architecture, Cloud-Native Applications, Service Orchestration.

ISSN: 3071-3498
PP: 01-12.

1. INTRODUCTION

1.1. Background

The design constraints of contemporary software systems have been radically overhauled by the speedy and unabated increase in data generation. Modern applications like big data analytics engines, social media application solutions, real-time monitoring systems, or e-commerce applications are required to operate in large volumes of data with a high level of availability, a minimal level of latency, and stable operation. The traditional monolithic designs used where systems components are highly bound into one deployable interface find it difficult to consider such demands. With the growth of data loads, scaling monolithic systems becomes complicated and ineffective, and in most cases, the whole application must be replicated or redistributed. Additionally, a tightly coupled component would be complicated to maintain since changes or failures in a single module will spread to other parts of the system leading to diminished system reliability and escalated downtime. The microservices architecture has proven to be an efficient solution to these deficiencies by splitting applications into a collection of tiny, independent services that interact via slim protocols. A microservice is also focused on a business capability and can be scaled and deployed flexibly because they are autonomous and thus can be developed on a case-by case basis. Such autonomy allows the organizations to scaled up only those services whose demand is high as opposed to the entire system, which results in optimal resource usage and performance. Besides, microservices are more isolative of faults, where failure of one service does not affect the whole application. Consequently, the microservices architecture is specifically the most applicable one in data-intensive applications, where heterogeneous workloads, various data access patterns, and unceasing evolution are typical.

1.2. Importance of Scalable Microservices Architecture

Fig 1 - Importance of Scalable Microservices Architecture

1.2.1. Handling Growing Data Volumes

Microservices architecture that is capable of being scaled is particularly essential in handling the ever-growing amount of data that is produced by the modern applications. Systems that are data-intensive tend to have an uneven and unpredictable workload, with some of the services receiving much more data than the rest. Microservices allow these components to arrive in scale and ensure

ISSN: 3071-3498
PP: 01-12.

effective processing and avoid performance bottlenecks. This scalability gives this targeted scalability and enables the systems to remain responsive as the volumes of data increases exponentially.

1.2.2. Improved System Performance and Responsiveness

Microservices architecture improves the overall system performance, as well as reduces response time by letting individual services be scaled according to demand. High traffic services are dynamically replicable and the requests can be processed parallely. This creates more efficient load balancing and user experience especially when there is the need to have real time data processing and low latency applications.

1.2.3. Enhanced Fault Isolation and Resilience

Microservice scalability is related to resiliency of a system. Service isolation means that system failure in a single component is not transferred to the whole system. Scalable deployments often default on redundancy where traffic can be rerouted to healthy service instances on failure. This is necessary in order to keep high levels of availability in data-driven and mission-critical applications.

1.2.4. Efficient Resource Utilization

Scalable microservices architectures enable the effective utilisation of computing resources through capacity allocation where the capacity is required. The resources are dynamically scaled by automated scaling mechanisms, and the level of over-provisioning and under-utilization is minimized. This is not only efficient in enhancing performance of the system but also, it also lowers the cost of running the system hence the architecture is appropriate with cloud-based environments.

1.2.5. Support for Continuous Evolution

Lastly, scalable microservices design enables uninterrupted growth and develop a system. Without affecting the whole application, independent services can be updated, optimized or replaced. This elasticity is essentially needed when operating data intensive systems that need to adapt quickly to the emerging needs, new technologies and new data processing requirements.

1.3. Architecture for Data-Intensive Applications

Data-intensive application architecture requires construction to be efficient in processing voluminous data with certain scalability, reliability and low-latency access. As opposed to traditional applications, where the main emphasis is on computation, data-intensive systems are characterized by the preeminence of issues regarding storage of data, data movement, and data processing. These applications invariably have a high level of data ingestion in real time over a large number of sources, real-time analytics, and intricate data operations that strain the system resources. With the increase in the amount, speed and spread of the data, the core architecture should be in a position to dynamically scale without affecting the performance or availability. The loose design of microservices allows it to be especially well adapted to data-intensive applications because of its decentralized and modular structure. The architecture enables services to scale to their respective

ISSN: 3071-3498
PP: 01-12.

data workloads by breaking down the system into independent services that can process a given dataset, or have a given data field. Decentralized data management, a data center ensures that every service has its own data store under its control, meaning that it can avoid contention and instead deploy heterogeneous storage technologies that are optimized to different forms of data and access patterns. The asynchronous communication models and event-driven communication models also help in achieving high-throughput processed data by decoupling the data producers and consumers so as to efficiently accommodate the streaming data and the batch data. There is also automated scaling, fault tolerance and monitoring which are important parts to architecture of data intensive applications. Horizontal scaling enables the system to achieve reaction to varying data loads and service isolation and redundancy results in reliability. Collectively, the architectural features allow the data-intensive applications to be high performance, resilient, and flexible in the contemporary a cloud environment.

2. LITERATURE SURVEY

2.1. Evolution of Microservices Architecture

The microservices architecture developed as a variant of the service-oriented architecture (SOA), due to the necessity of more agility, scalability, and the independence of software components. Microservices contrast with traditional SOA, which is often based on heavyweight middleware and coarse-grained services communicating with each other using heavyweight protocols, such as REST or messaging queues. The initial studies focused mainly studied the advantages of microservices in relation to the modularity of applications, isolating of faults, and the expedited process of development. After the proliferation of cloud computing and containerization technologies, microservices became a popular style of architecture of large-scale distributed systems. Nevertheless, a significant part of the preliminary work was on the topic of functional decomposition and service separability with little consideration to the challenges posed by data-intensive workloads.

2.2. Data Management in Microservices

One of the greatest issues in microservices based systems is data management since the services are distributed across the board. Various researches promote the principle of database-per-service, according to which the microservice has a data store of its own in order not to be tightly coupled but to evolve as an independent entity. Although this strategy improves scaling and autonomy of services, it makes the data consistency and control of transactions across the service more complex. Classical ACID-conformant distributed transactions are frequently shunned as performance and scalability constraints. Consequently, alternative solutions to it have been suggested by other researchers, including eventual consistency models, event-driven communication and event sourcing patterns. The purpose of these approaches is to ensure data reliability at the cost of temporary inconsistencies to achieve better scalability and resilience.

2.3. Scalability and Performance Studies

The idea of scalability and performance has been used to focus on the research on microservices and especially when comparing it to monolithic architecture. Empirical research points

ISSN: 3071-3498
PP: 01-12.

to the fact that microservices may serve better than monolithic systems when subjected to variable and unpredictable workloads because it is able to scale individual services separately. The container orchestration platforms can be used to provide the horizontal scale, load balancing being automated, and resource usage being efficient. Although these have its benefits, studies also point out the performance overheads accrued by microservices particularly in respect to higher inter-service communication and network latency. The problem of over-service fragmentation and synchronous communication pattern might have adverse effects on response time thus performance optimization is an important aspect in design.

2.4. Research Gaps

Although the background research of microservices architecture is rather significant, there are many gaps even in the case of data-intensive applications. The existing research usually concentrates on the infrastructure level and is concerned with the strategies of scalability and deployment but gives minimal instructions about how to optimize data flow, and service interaction patterns. It has no single architectural structures that can be used both to establish the issue of computational scalability and to manage the data cleanly as well. Further an issue is that most of the proposed solutions fail to provide sufficient information on the performance implication of complex data dependencies between services. These restrictions underscore the necessity to consider a comprehensive strategy that includes both the scalable infrastructure architecture and optimized data handling and communication of services that ensure the encouragement of the proposed methodology.

3. METHODOLOGY

3.1. Architectural Design Principles

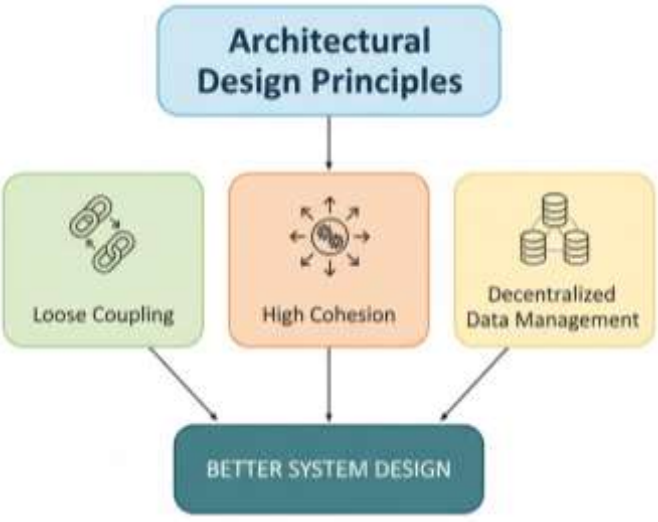

Fig 2 - Architectural Design Principles

ISSN: 3071-3498
PP: 01-12.

3.1.1. Loose Coupling

One of the principles of the proposed architecture is loose coupling where the services communicate not by depending on each other, but by well-defined APIs and asynchronous message queues. This model enables the development, deployment and scaling of individual services without causing the impact on other elements. The inter-service dependencies can be reduced, making the system more robust to any failures and simpler to upgrade as time progresses which is particularly crucial in the dynamic and distributed cloud setup.

3.1.2. High Cohesion

The high cohesion is ensured because each microservice is designed to carry out one and clear business activity. This explicit segregation of functions enhances code maintainability, and readability and, in minimizing the complexity of the system. Cohesive services are simpler to test and adjust, allowing them to be developed quicker and their updating to be more dependable. Moreover, high cohesion is helpful in isolating faults, since the troubles of a particular service are not highly likely to spread throughout the network.

3.1.3. Decentralized Data Management

The architecture implements decentralized data management in which the data ownership is delegated to the specific services instead of using a centralized database. The services have independent data stores, each of which minimizes interconnection in the data layer and allows each to independently support its workload depending on the service load. Despite the difficulties associated with data consistency posed by this approach, it improves autonomy and conforms to the notion of microservices since it promotes matter, scalability, and autonomous development of services.

3.2. Layered Microservices Architecture

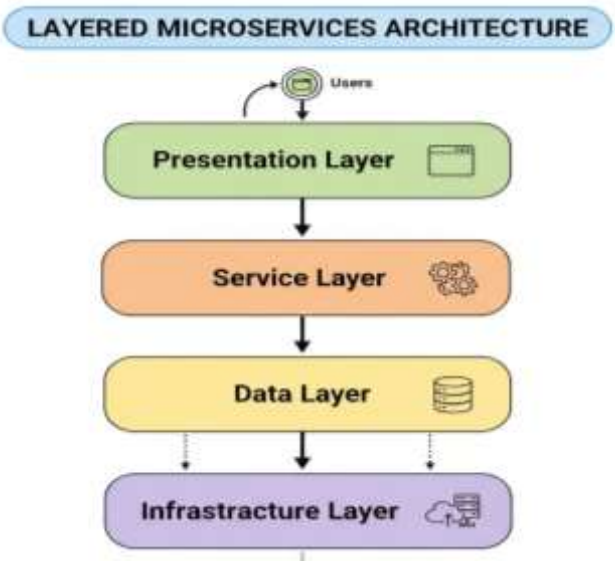

Fig 3 - Layered Microservices Architecture

ISSN: 3071-3498
PP: 01-12.

3.2.1. Presentation Layer

The presentation layer is the one that deals with all the interactions that the user has with the system and also through which they interact with the outside world. It has user interfaces, client applications and API gateways that redirect requests to relevant backend services. The concerns included in this layer are the validation of the requests, authentication as well as the load balancing, securing and efficient access to the underlying microservices and the abstraction between the user and the internal system components.

3.2.2. Service Layer

The service layer includes the main microservices which execute the business logic of the system. The operations of each microservice are independent and oriented towards a functional responsibility. The services interact with each other on a light weight protocols or messaging systems, to be able to scale, and be flexible. This layer is the foundation of the application and handles the processing of requests, implementation of business regulations and also workflows across the system.

3.2.3. Data Layer

The data layer ensures the storage and retrieval of data through distributed databases and storage systems which are specific to service demands. Each microservice has its own data store, which contributes to supporting the decentralization of data management and consists of minimizing inter-service relationships. It allows scaling independence and heterogeneity of technology, which allow high volumes of data. Reliability and the eventual consistency can frequently be enforced using mechanisms like replication, and the use of eventual consistency.

3.2.4. Infrastructure Layer

The infrastructure layer offers the environment basis needed to interact and run microservices. It incorporates containerization technology and orchestration platforms enabling the automation of the deployment of services, scaling and monitoring. The layer provides the system with the high availability, fault tolerance, and efficient resource utilization to meet the needs of the system to act dynamically in response to workload changes without causing operational instability.

3.3. Communication Model

By balancing among scalability, responsiveness, and reliability, the proposed architecture uses an alternative model of communication: a hybrid approach based on asynchronous messaging and synchronous interactions based on the REST protocol. Message brokers are the most common platforms to implement asynchronous communication and this provides services with the ability to exchange information through events without any direct dependencies or immediate response. This will greatly decrease the time dependence of the services so that they can time and again work on their own and they can go on running even in the situations when other services are unavailable at all. Asynchronous messaging enables producers of data to be decoupled with consumers and improves the behavior of systems, including fault tolerance and high throughput processing of data, which is especially useful to data-intensive and large-scale distributed applications. The system has a

ISSN: 3071-3498
PP: 01-12.

synchronous method of communication as well as asynchronous messaging like REST based API in situations which demand instant response like user driven requests or on-the-fly verification. RESTful communication offers a interaction interface that is simple and standard, and therefore it is appropriate in requestsresponse patterns, where latency is strictly important. Nevertheless, synchronous calls are restricted a few times to avoid over-coupling and performance crashes, whereas non-critical processes are asynchronized and run on events as processes are difficult and costly to time. Event-driven messaging model allows services to publish events when some significant changes of state happen and this can be consumed by various services as required. This design is scalable because it does not require changes to be made to the existing components in order to enable new services to subscribe to events. Load leveling is also achieved through message brokers since effective load brokering would ensure that services are not overwhelmed by sudden rises in workload. The interaction of synchronous and asynchronous communication combined ensures flexibility of service interaction, and performance under varying workloads is optimized, and the combination meets microservices best practices which encourage loose coupling, scalability and resource efficiency.

3.4. Scalability Mechanism

Horizontal scalability is a fundamental use case in the proposed architecture as it ensures a steady performance even in the circumstances of different and unpredictable workload. Horizontal scaling is done by increasing or reducing the number of service instances in real-time instead of improving the capacity of an instance. This is especially suited to systems that are built using microservices because each microservice is able to scale on its own resource needs. Auto scaling policies are established based on real-time workload metrics which include CPU utilization, memory utilization and request latencies which enables the system to respond efficiently to demand fluctuations. My monitored metrics are checked against predetermined limits, which causes new service instances to be created to spread the incoming traffic more uniformly and therefore reduce response time and avoid service degradation. On the other hand, in case of a decline in workload, unneeded cases get dismissed to maximize the use of resources as well as costs incurred in operations. Such elasticity allows the system to ensure its performance efficiency and take care of over-provisioning of resources. The scaling decisions are taken accurately along with real load of the services as their quantification is fined down to the fine-grained metrics. Another metric that is of specific significance to the auto-scaling strategy is request latency that is the immediate measure of user experience and system responsiveness. The architecture ensures that bottlenecks in performance are handled by having latency-based scaling triggers and using the CPU utilization in the manner in which bottlenecks in performance are taken. Moreover, horizontal scalability enhances fault tolerance because failures in one instance do not affect it: a failed instance will automatically be rerouted to other inactive instances. By and large, scalability mechanism can promote the reliability of the system, high-availability, and allow the architecture to manage the growth and visage of work loads in a cloud-based environment effectively.

WCRG
Global Research
ISSN: 3071-3498
PP: 01-12.

4. RESULTS AND DISCUSSION

4.1. Performance Evaluation

Table 1: Performance Evaluation

Metric	Percentage Change (%)
Avg. Latency	50% reduction
Throughput	105.9% increase

4.1.1. Average Latency Reduction (50%)

The experimental findings indicate that the average latency is reduced by half when the proposed microservices architecture is used in comparison to the monolithic system. This has been reduced a lot thus showing that the system can scale up under heavy loads of requests. The scaling of independent services, the optimization of resources allocation, and the minimization of processing bottlenecks may be explained with the decrease of latency. The architecture reduces request waiting durations through the distribution of workloads among different instances of a service to the end of reducing the total responsiveness of the architecture hence increasing the user experience.

4.1.2. Throughput Improvement (105.9%)

The suggested architecture shows that the increase in throughput is 105.9% higher than in the monolithic paradigm, which means that the proposed architecture will be able to support over twice as many requests per second as the previous one. This enhancement brings out the power of scaling horizontally and supporting parallel request processing made possible by the microservices design. The auto-scaling mechanisms and independent service deployment of the system enable it to handle large volumes of traffic without degradation of performances. The architecture is therefore ideal in applications with data-intensive and heavy demand where maintained performance in the face of large workloads is vital.

4.2. Fault Tolerance Analysis

Another important feature of the suggested microservices architecture is a fault tolerance that guarantees the reliability of the system and its active existence even under the circumstances of partial failures. The architecture carries a service isolation principle, involving the functioning of different microservices as independent entities that exist in a self-executing environment. This isolation ensures that failure of any of the services does not affect the others directly thus limiting faults and ensuring system stability. With a service failure or a service becoming unavailable, the rest of the services are capable of being allowed to carry on with normal operation of the system as the services may gracefully degrade, instead of having a total shutdown. Automated recovery systems are in the center stage to reduce downtimes and enhance system resilience. The service failures are identified at the time through health checks and constant monitoring. On detecting a failure, orchestration platforms will automatically re initialize failed service instances or give them new ones, which do not need manual intervention. The load balancers dynamically redirect traffic between failed instances so that services remain accessible. These self- recovery features greatly decrease the mean time to recover as well as improve system reliability. Fault tolerance is further achieved by

ISSN: 3071-3498
PP: 01-12.

having redundancy and replication of critical services. Several clones of the same service are kept on various nodes thus removing the possibility of single points of failure. Asynchronous communication and event based messaging are also relevant to resilience as they enable the services to keep on its tasks without any further dependence on down-stream services that might be unavailable at the moment. Altogether, the proposed architecture can be very high in the availability and very powerful in the fault tolerance due to service isolation, automated recovery, and redundancy which guaranteed that it can be applied to large-scale and data-intensive applications in the dynamic cloud environments.

4.3. Discussion

The results of this research support the idea that microservices architecture is an important tool regarding scalability and resilience, especially in regard to applications of considerable data load that need to be supplied under dynamic load variations. The experimental results prove that the suggested architecture is effective in supporting the growing data volume and request rate using independent scaling of services and decentralized management of data. In contrast to monolithic systems, which tend to become bottlenecks during performance due to the workload, microservices enable separate components to increase in load depending on the demand, which leads to harder throughput and smaller response time. This architecture facilitates microservices to be particularly appropriate in any cloud-based setting where workload patterns are random. The architecture is also highly resilient with regard to service isolation and automated recovery processes besides the virtue of scalability. The individual service failure does not disrupt the system cause discontinuities because the failures are confined to individual services and do not have an impact on the system. The operation reliability is promoted greatly by the possibility of automatic detection of failures and recovery or services without human interference. Such aspects are especially significant in data-intensive applications where constant availability and steady performance are needed. Nevertheless, the article also points out the drawback of the microservices that is enhanced architectural and operational complexity. The service design must be done with care, lest there should be too much service fragmentation resulting into overheads of communications and latency. Besides, distributed services are important to manage well in light of failure diagnosis and keeping performance due to proper monitoring and observability. Cascading failures and resource ineffectiveness are some of the problems that may not be noticed without extensive monitoring. As a result, despite the benefits of microservices architecture in terms of scalability and resilience, its effective implementation requires a carefully designed approach, effective monitoring plans, and well-conducted operational procedures to maximize its potential.

5. CONCLUSION

In this paper, a scalable microservices architecture that is specifically aimed at providing the needs of data-intensive applications in contemporary distributed computing setup has been presented. The suggested solution will be based on the existing concept of microservices but will reassign it with architecture methods that focus on the decentralized nature of data control,

ISSN: 3071-3498
PP: 01-12.

asynchronous communication, and horizontal scaling through automatism. The architecture creates less tight integration of the data layer by giving data ownership to each service and also allows the service evolution to occur independently. This architecture will provide flexibility and scalability, which is necessary to process large volumes of data and rapidly evolving workloads. There is also a performance and resilience of the system as asynchronous communication is integrated into it, with the help of event-driven messaging. The architecture avoids dependency related failures by decoupling services interactions both in time and implementation, providing better throughput in terms of heavy loads. Meanwhile, the selective usage of synchronous communication with the use of REST makes user-facing operations responsive. The system can also perform real-time workload measurements which are used to automate scaling mechanisms to dynamically tune resource allocation to provide optimal performance at a cost-efficient price. All these design aspects directly solve the typical problems with distributed systems, such as performance xenophobia, fault propagation, and inefficient use of resources. The feasibility of the proposed architecture can be confirmed with the help of experimental evaluation that indicates that this type of architecture can help to significantly reduce the throughput and response time in relation to the time-honored monolithic systems. The performance at load patterns is also emphasized in the outcome, showing the capability of the architecture to maintain high performance with regeneration of data load, confirming its advantages with huge-scale, data-sensitive applications. Moreover, increased fault tolerance and automatic recovery help to achieve greater levels of availability and operational reliability, which are fundamental in cloud-based migrations. In spite of such benefits, the study recognizes the fact that micro-services systems add the complexity of service coordination, monitoring, and maintaining data consistency. Therefore, the benefits of this approach would require careful design of the architectural systems and powerful mechanisms of observability in order to achieve the full benefits of this approach. Future directions include developing models of more advanced concepts of data consistency that have good performance in a distributed environment and are also correct. Moreover, it is a good way to perform predictive scaling and optimization of the existing resources using the techniques of artificial intelligence and machine learning and become a promising direction to make the systems even more scalable and efficient, as well as adaptive.

REFERENCES

[1] Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media.
[2] Lewis, J., & Fowler, M. (2014). Microservices: A definition of this new architectural term. *Martin Fowler Blog*.
[3] Dragoni, N., Lanese, I., Larsen, S. T., Mazzara, M., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, today, and tomorrow. *Present and Ulterior Software Engineering*, Springer.
[4] Pautasso, C., Zimmermann, O., & Leymann, F. (2017). Microservices in practice, part 1: Reality check and service design. *IEEE Software*, 34(1), 91–98.
[5] Villamizar, M., Garcés, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., Casallas, R., & Gil, S. (2016). Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. *IEEE Cloud Computing*, 3(6), 10–18.
[6] Taibi, D., Lenarduzzi, V., & Pahl, C. (2018). Architectural patterns for microservices: A systematic mapping study. *Proceedings of the 8th International Conference on Cloud Computing and Services Science*.
[7] Fowler, M. (2017). Event sourcing. *Martin Fowler Blog*.
[8] Richardson, C. (2018). *Microservices Patterns: With Examples in Java*. Manning Publications.

ISSN: 3071-3498
PP: 01-12.

[9] Namiot, D., & Sneps-Sneppe, M. (2014). On micro-services architecture. *International Journal of Open Information Technologies*, 2(9), 24–27.

[10] Zhang, Q., Chen, M., Li, L., & Li, M. (2018). Research on data consistency in microservice architecture. *IEEE International Conference on Software Engineering and Service Science*.

[11] Pahl, C. (2015). Containerization and the PaaS cloud. *IEEE Cloud Computing*, 2(3), 24–31.

[12] Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software*, 35(3), 24–35.

[13] Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: A distributed messaging system for log processing. *Proceedings of the NetDB Conference*.

[14] Chen, L. (2018). Microservices: Architecting for continuous delivery and DevOps. *IEEE International Conference on Software Architecture (ICSA)*.

[15] Zhou, M., Li, L., & Chen, J. (2019). Performance evaluation of microservices architecture using containers. *Journal of Cloud Computing*, 8(1), 1–14.