A comprehensive analysis of Microservices

[1]Mohummed Washeem, [1]Dr Tarun Maini, [1]Sushant Jhingran, [1]Gourishankar Mishra, [1]Pradeep Kumar Mishra.

[1]Department of Computer Science Sharda University Knowledge Park III, Greater Noida, Uttar Pradesh 201310, India

E-mail:
[1]Shaikhwasheem@gmail.com,[1]Taruniitbhu@gmail.com,[1]Sushantjhingran@gmail.com,[1]Gourisankar.mishra @sharda.ac.in,[1]Pradeepkumar.mishra@sharda.ac.in

Abstract.

People now need applications that are quick, efficient, and dependable. Only the programmer can offer characteristics like dependability and efficiency if the customer has a high speed internet connection and needs the client's app. Therefore, choosing the appropriate software architecture is essential before implementing the functionalities that have been approved for the project. Monolithic and microservice architectures are the most often used designs; both may achieve the same result but have different advantages and disadvantages. Cloud computing provides plentiful opportunities to the clients or companies to create design and personalize the business application online. Nowadays many companies are deploying their applications on cloud computing environments like infrastructure as a service (IaaS) platform as a service (PaaS) and software as a service (SaaS). Deploying this services on monolithic architecture creates difficulties to handle their number of modules. Using microservices in the development of the application makes it easy to build, deploy and maintain the application by its various features like multi code base, independency of technology, lower cost and increased efficiency etc. The modularity and reduced size of microservices allow flexibility that is helpful to both operational and development teams. This study aims to identify and the representation of the deployment and challenges in microservices.

Keywords: Microservices, Monolithic, DevOps.

DOI Number: 10.48047/NQ.2022.20.20.NQ109300 NeuroQuantology2022;20(20):3030-3038

1 Introduction

Microservices acquires a lot of popularity of the industrial and researchers in modern days by its effective functional characteristic. Microservices are flexible and efficient approach to building operative application. Microservices breaks the large application in smaller parts that work independently of each other services, each runs in its own process and communicates over well-defined lightweight protocol such as HTTP or MQTT [1], [2]. Since monolithic application is a single code base application that faces difficulties to adopt new technologies, that's why for a single change it need to be redeployed the whole application. To overcome these problems many companies started relocating theremonolithic applications to microservices architecture. The very basic concept of the microservices is its breaks down the applications into essentially multiple smaller independent services. Breaking the services in smaller part makes it easy to build and alsoimprove the productivity of microservices. As microservices are combination of smaller services that provides the ability to operate the whole applications simultaneously.

Development, deployment and maintenance of the microservices are done separately. Which gives the ability of choosing the best suitable technology to the development team to find the current needs of business behavior [3]. Microservices architecture allows the development team to develop, test, deploy and update its services without interrupting other services. Which makes it easy and faster development. As every services can be written in a different languages makes microservices more scalable. Microservices are primarily utilized in the service sector due to their lightweight design and dynamic deployment [9].

The new DevOps (Development and Operations) culture is compatible with microservices as well. With the use of techniques like automation, continuous delivery, and continuous integration, DevOps focuses on reducing the time between development and deployment [6].

1.1 Monolithic Architecture

The conventional unified model for the construction of a software application is called a monolithic architecture. Monolithic here refers to something that is "made entirely of one piece." The term monolithic can also indicate "too huge" and "unable to be changed," according to the Cambridge Dictionary. Software developers have employed monolithic architecture successfully for the past few years. A monolithic application is a piece of software made up of various parts from a single platform [4].

Any software programmed whose modules cannot be run separately is referred to as a monolith [8]. An illustration of a monolithic application that provides business logic for e-commerce may be seen in the image below.

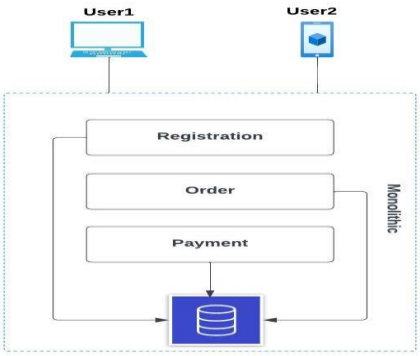

Fig. 1. Monolithic Architecture

1.2 Microservice Architecture

The old monolithic design is no longer the ideal option due to rising complexity and the requirement for highly scalable and reliable applications. After a certain point, the monolithic architecture frequently inhibits the application's performance and scalability. An application architecture known as a "microservices architecture" is one in which the application is created as a group of services. Microservices are a programmed architectural approach, not an extra communication layer [15]. It offers the framework for autonomously creating, deploying, and maintaining microservices architectural diagrams and services. One microservice to another may use a different language and database. Instead of exchanging data, they communicate with one another via the Representational State Transfer (REST) protocol. Agility, autonomy, scalability, robustness, and simple continuous deployment are the most significant advantages of employing microservices [3]. Microservices architecture is a common method of organizing software systems is through the use of loosely couple services. The basic objective of microservices is to manage and maintain each individual component of an application.

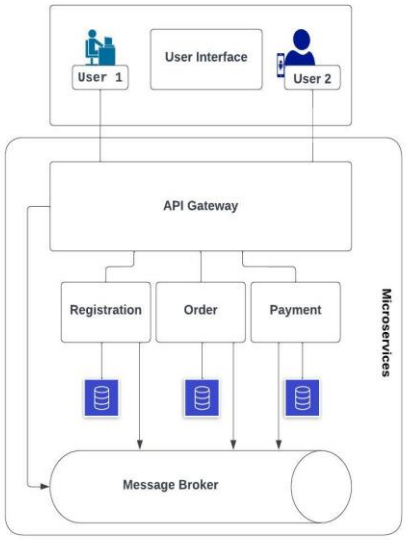

Fig. 2. Microservices Architecture

The microservices' architecture is visible in the above figure. Different sorts of users can interact with microservices through user applications with the aid of API gateways. Every microservice module has its own database, and this design uses a message broker for asynchronous communication between microservices. Microservices, which are organized around business capabilities and have smaller code bases than conventional monoliths, claim to be more maintainable than these latter [16].

The below image shows the development of microservices according to Google Trends between 2004 and 2021.

Fig. 3. Google Trends statistics for searching word "microservices" between 2004-2021 [4]

2 Background

The Service-Oriented Architectural (SOA) gave rise to the Microservice architecture paradigm . Although SOA services also have specific duties, they are not autonomous. In such an architecture, the services cannot be individually turned on or off. This is because none of the separate services are truly autonomous or full stack. Services in SOA are unable to be deployed individually [5]. The architecture of the entire system is drastically altered. Several Microservices are split apart from a single monolithic service. As a result, adjustments are made to both the environment's requirements and the internal architecture of the service. A full environment is needed for each microservice, which may be thought of as a full stack [5].

2.1 Spring Boot

Based on the Spring framework, Spring Boot is a framework. With the help of this tool, a programmer may create web apps more quickly and easily. Spring Boot practically doesn't need configuration, in contrast to the Spring framework. More XML descriptors are now required to be provided. You just need to pick the dependencies you wish to employ carefully [4]. For creating microservices for mobile apps and web applications, developers have a variety of reasons why they select Spring Boot. To build the source language, Spring Boot employs Boot Initializer. Boot Initializer is used by Spring Boot to build the source language. Users may conserve space on their devices and quickly load applications with the help of to this bootstrapping mechanism.

2.2 Docker

A technology known as containerization organizes system libraries, dependencies, and applications into a container-like structure. Applications that have been developed and arranged can be deployed and executed in a container. Docker, a platform, ensures that the application

functions in all environments. Additionally, it automates the deployment of apps into containers. The container environment in which the applications are run and virtualized is supplemented by Docker with an additional layer of deployment engine. Docker contributes to the creation of a rapid and light environment that can run code effectively. Docker's four key components are its containers, clients/servers, images, and engine [4].

Docker is a well-liked container system that is backed by numerous tools, including Kubernetes. Because DockerHub, its registry service, is readily available, it is reasonably simple to copy Docker instances [22]. For the deployment of an application, numerous parameters, dependencies, and other requirements are always required. Utilizing the Docker platform to develop, distribute, and manage applications using containers is a wise precaution to take in order to avoid this. The process of installing software in Linux containers, known as containerization, has a number of advantages, such as flexibility, lightness, interchangeability, and portability [4].

2.3 API Gateway

The client apps in a microservices architecture typically need to use functionality from many microservices. The client must manage several calls to microservice endpoints if such consumption is carried out directly. Redirect or route requests (layer 7 routing, often HTTP requests) to the endpoints of the internal microservices using the reverse proxy provided by the API Gateway. The gateway gives the client apps a single endpoint or URL and then internally routes the requests to a collection of internal microservices. The main responsibility of the API gateway is to always direct incoming requests to the appropriate downstream services. It may also perform protocol translation (i.e., between web-friendly protocols like HTTP and WebSocket and internal ones like AMQP and Thrift binary RPC) and occasionally compose requests [18]. In cloud computing, JavaScript Object Notation (JSON) and XML are the most widely used data types[21].

2.4 DevOps

Microservices and DevOps are two terms that are frequently used together. A group of practices called DevOps tries to integrate software development methods with deployment and operations methods. DevOps is a collection of procedures that aims to speed up the process of altering a system and implementing that change in a live setting [7]. Microservices and DevOps are dependent on one another in several ways. They both rely significantly on virtualization and the cloud [6]. The driving drivers behind microservice deployments are DevOps techniques like Continuous Integration and Continuous Delivery. The DevOps methodology is crucial to the microservices architecture because it enables developers and IT operations to collaborate closely and produce higher-quality software much more quickly.

3 Challenges in Microservices

3.1 Intercommunication

Monitoring, tracking, and related activities. That's obviously crucial when you start using microservices since communication inside an application is not done by calling a function that is running in memory. A network call may or may not be made locally on the system in this case. It is crucial to be able to see, follow, and trace conversations across an application. Without it, you're effectively wearing a blindfold and attempting to spot a black cat in a pitch-black room.

Changing the communication mechanism is one of the main challenges when converting to microservices-based applications. because microservices are dispersed and interact with one another via network-level inter-service

communication. REST and HTTP communication are cited in several studies as characteristics of microservices, however this isn't always the case. REST APIs are among the most widely used technologies for microservices, although there are other possibilities as well, such RPCs [6].

Synchronous communication- For returning a sync response during synchronous communication, HTTP or RPC protocol is used. The customer submits a request and waits for the service to respond. Therefore, client code must block its thread until the server's response arrives [1].

Asynchronous communication- The client sends a request but does not wait for a response from the service in asynchronous communication. Therefore, the crucial issue is that the client shouldn't have blocked a thread while it was waiting for a response. It is considered to Use message queues rather than the REST request/response structure with the asynchronous messaging microservice design template [13]. The most widely used asynchronous communications protocol is AMQP (Advanced Message Queuing Protocol) [1].

3.2 Scale up and Scale down in real time

However, scaling to meet demand is a major problem for any IT implementation, and microservices are no different. A separate instance of the type may have different loads on the various microservices. Even though stateless systems are where microservices are frequently implemented, they are not inherently scalable [20]. Your microservice should automatically scale down as well as up. It brings down the price of the microservices. We are able to dynamically spread the burden. Each individual microservice is the unit of scalability for microservices. Services can therefore be expanded differently at runtime depending on their unique needs [12].

3.3 Load Balancing

Sharing incoming network traffic in real-time or on demand across servers in a server farm or server pool is known as load balancing. This sharing procedure may be carried out uniformly or in accordance with predetermined guidelines. Round Robin, Least Connections, and other rules. The majority of load balancing issues arise as resource populations shift and infrastructure expands to meet rising demand. In these circumstances, if load is not distributed equally among the nodes, the end-user experience may suffer, and application performance may finally deteriorate. In server-side load balancing, the user depends on the load balancer to select the best instance of the target service to which requests should be sent. Numerous microservices located behind the load balancer handle user requests [1].

3.4 Architectural complexity

Independently creating and deploying microservices adds complexity to many processes including continuous development and delivery, monitoring, scalability, and recovery. Therefore, manually managing microservices is not practical in general [1]. By not sharing libraries, each microservice has an infrastructure that is completely independent of every other microservice [10]. The programming language that provides the most sense for the result should be used to create microservices. It is not necessary for them to be created in the same programming language because they work together to create a complex application [19].

3.5 Testing

Scale issues contribute to the fact that standard testing isn't always effective for microservices. Numerous separate services may not be available for testing at the same time for apps based on microservices. Similar to how it might be challenging to test things "the old-fashioned way," microservices are all put

together at the conclusion of a project thus, alternatives are definitely a smart idea. Since a single user's activity may start a chain reaction involving several microservices that are in communication with one another, it might be challenging to isolate the originating issue in some cases. Initially, testing a microservice is simpler than testing a monolith. However, based on the number of components and links between them, integration testing may be considerably more difficult [11]. A microservice should comprise technical endpoints used for inter-microservice communication in addition to endpoints for the execution of business logic. It is necessary to prepare test data in a specified manner in order to test each endpoint. There could be other endpoints that need testing.

3.6 Container image vulnerability:

There is frequently a requirement for the deployment of new services and updated versions of current services due to the fact that microservices have the concept of collaboration in large-scale systems. technology based on containers (like Docker, Among the most widely used technologies are Kubernetes, etc. Encouraged to be used to deploy microservices, black-box repurposing to enable commercially available installations[1]. Services are highly decoupled, so each one can grow independently by simply starting and stopping a number of copies, or containers, of the same image [14]. A Docker container typically only has partial access to host resources, but complete access can be granted by establishing the container as a "privileged" container [17].

3.7 Lack of Standardization.

Microservices are established independently, thus there is no one solution that works for all of them. Your microservices may wind up being unevenly designed and implemented as a result, which will make it more challenging to maintain over time and difficult for new team members to understand your application. Establishing standards for your microservices is one way to address this problem. This includes coding standards, directory layouts, and communication protocols. Having a set of standards may help your microservices be more consistent, readable, and maintainable.

4 Result

The proposed paper covers the key features of the microservices. The deployment aspect, the difficulties with microservices, and a comparison between monolithic design and microservices are all explored in this article. The comprehension of the benefits and drawbacks is provided below after literature of the selected article.

4.1 Advantages and Disadvantages of Microservices

Advantages

- One of the most important advantages of microservices is scalability. Increases in user traffic have an impact on all areas of a monolithic programmed.
- Because of the programming language and technology agnosticism, developers may connect microservices that are written in any language and operating on any platform.
- Total costs could be lower since microservices are usually simpler and more efficient than monolithic applications. Due to their independence, microservices don't require as much coordination and communication as monolithic applications do.
- One of the key benefits of microservices is the capacity for businesses to take a more granular approach to data protection. Due to the fact that each service is responsible for a distinct job, it is easier to implement security measures at the service level.

Disadvantages

- The higher level of complexity may be a major challenge for organisations that are not used to working with microservices. Additionally, because microservices are so autonomous, it could be difficult to find and address problems.
- Microservices heavily rely on the network to communicate with one another because they are designed to be autonomous. This might lead to slower networks and more network traffic.
- Due to their complexity and increased coordination needs, microservices also take longer to create than monolithic apps.
- Microservices have a limited potential for code reuse since they are usually developed using a range of programming languages and technological stacks. This can lead to longer development times and higher development expenses.

5 Conclusion

A distributed design strategy microservices architecture aims to get beyond the drawbacks of conventional monolithic structures. Microservices speed up cycle times while enabling the scalability of businesses and applications. Microservices are becoming more and more popular. Big businesses have adopted them and are now the market leaders in innovation. For beginners, the main goal of microservices is to divide huge software systems into smaller subsystems that are crucial activities that can be independently distributed. In this paper, we made an effort to list and go over the key points of contention around microservices. The architectural comparison between monolithic and microservices was the first step, followed by a discussion of inter-microservice communication and additionally, we covered some of its methods and frameworks including Docker and Spring Boot.

Finally, we came up with a few benefits and drawbacks of microservices. Additionally, we have made an effort to pinpoint the primary tools and procedures used to manage microservices.

References

[1] Isil Karabey Aksakalli , Turgay Celik , Ahmet Burak Can & Bedir Tekienerdogan (2021). Deployment and communication patterns in microservice architectures: A systematic literature review. The Journal of Systems & Software 180 (2021) 111014.

[2] Vladimir Yussupov , Uwe Breitenducher , Christoph Krieger , Frank Leymann Jacopo Soldani & Michael Wurster (2020). Pattern-based Modelling, Integration, and Deployment of Microservice Architectures. 2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC).

[3] Hulya Vural , Murat Koyuncu & Sinem Guney (2017). A Systematic Literature Review on Microservices . Conference Paper in Lecture Notes in Computer Science · July 2017.

[4] Konrad Gos ,Wojciech Zabierowski (2020). The Comparison of Microservice and Monolithic Architecture . 2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH).

[5] Florian Auer Valentina Lenarduzzi Michael Felderer Davide Taibi. From monolithic systems to Microservices: An assessment framework . Information and Software Technology 137 (2021) 106600

[6] Mohammad Sadegh Hamzehloui, Shamsul Sahibuddin, and Ardavan Ashabi. A Study on the Most Prominent Areas of Research in Microservices. International Journal of Machine Learning and Computing, Vol. 9, No. 2, April 2019.

[7] Balalaie, A., Heydarnoori, A. and Jamshidi, P., 2016. Microservices architecture enables devops: Migration to a

cloud-native architecture. *Ieee Software*, *33*(3), pp.42-52.

[8] Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R. and Safina, L., 2017. Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering*, pp.195-216.

[9] Jhingran, S. and Rakesh, N., 2021, July. Performance factor impacting behavior of microservices in various hosting domains. In *2021 Fourth International Conference on Computational Intelligence and Communication Technologies (CCICT)* (pp. 160-164). IEEE.

[10] Pacheco, V.F., 2018. *Microservice Patterns and Best Practices: Explore patterns like CQRS and event sourcing to create scalable, maintainable, and testable microservices*. Packt Publishing Ltd.

[11] Larrucea, X., Santamaria, I., Colomo-Palacios, R. and Ebert, C., 2018. Microservices. *IEEE Software*, *35*(3), pp.96-100.

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

[13] Pahl, C. and Jamshidi, P., 2016. Microservices: A Systematic Mapping Study. *CLOSER (1)*, pp.137-146.

[14] Butzin, B., Golatowski, F. and Timmermann, D., 2016, September. Microservices approach for the internet of things. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)* (pp. 1-6). IEEE.

[15] Shadija, D., Rezai, M. and Hill, R., 2017, September. Towards an understanding of microservices. In *2017 23rd International Conference on Automation and Computing (ICAC)* (pp. 1-6). IEEE.

[16] Knoche, H. and Hasselbring, W., 2018. Using microservices for legacy software modernization. *IEEE Software*, *35*(3), pp.44-49.

[17] Amaral, M., Polo, J., Carrera, D., Mohomed, I., Unuvar, M. and Steinder, M., 2015, September. Performance evaluation of microservices architectures using containers. In *2015 ieee 14th international symposium on network computing and applications* (pp. 27-34). IEEE.

[18] Chandramouli, R., 2019. Microservices-based application systems. *NIST Special Publication*, *800*(204), pp.800-204.

[19] Bakshi, K., 2017, March. Microservices-based software architecture and approaches. In *2017 IEEE aerospace conference* (pp. 1-8). IEEE.

[20] De Lauretis, L., 2019, October. From monolithic architecture to microservices architecture. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)* (pp. 93-96). IEEE.

[21] Sill, A., 2016. The design and architecture of microservices. *IEEE Cloud Computing*, *3*(5), pp.76-80.

[22] Nehme, A., Jesus, V., Mahbub, K. and Abdallah, A., 2019. Securing microservices. *IT Professional*, *21*(1), pp.42-49.

[23] Daya, S., Van Duy, N., Eati, K., Ferreira, C.M., Glozic, D., Gucer, V., Gupta, M., Joshi, S., Lampkin, V., Martins, M. and Narain, S., 2016. *Microservices from theory to practice: creating applications in IBM Bluemix using the microservices approach*. IBM Redbooks.

[24] Chandramouli, R., 2019. Microservices-based application systems. *NIST Special Publication*, *800*(204), pp.800-204.

[25] De Lauretis, L., 2019, October. From monolithic architecture to microservices architecture. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)* (pp. 93-96). IEEE.

[26] Salah, T., Zemerly, M.J., Yeun, C.Y., Al-Qutayri, M. and Al-Hammadi, Y., 2016, December. The evolution of distributed systems towards microservices architecture. In *2016 11th International*

Conference for Internet Technology and Secured Transactions (ICITST) (pp. 318-325). IEEE.

[27] Brown, K. and Woolf, B., 2016, October. Implementation patterns for microservices architectures. In *Proceedings of the 23rd Conference on Pattern Languages of Programs* (pp. 1-35).

[28] Di Francesco, P., 2017, April. Architecting microservices. In *2017 IEEE international conference on software architecture workshops (ICSAW)* (pp. 224-229). IEEE.

[29] Di Francesco, P., Lago, P. and Malavolta, I., 2019. Architecting with microservices: A systematic mapping study. *Journal of Systems and Software*, *150*, pp.77-97.

[30] Fetzer, C., 2016. Building critical applications using microservices. *IEEE Security & Privacy*, *14*(6), pp.86-89.

[31] Pimentel, E., Pereira, W., Maia, P.H.M. and Cortés, M.I., 2021, May. Self-Adaptive Microservice-based Systems-Landscape and Research Opportunities. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)* (pp. 167-178). IEEE.