**International Conference of Sciences, Engineering & Environmental Technology (ICONSEET), 5(19): 148 – 156, 2020**
**ISSN 0794-9650**

www.repcomseet.com

# MICROSERVICES ARCHITECTURE: A BETTER OPTION OVER MONOLITHIC ARCHITECTURE

**Lawal Moshood O.[1,*], Ileladewa Adeoye A.[2], Lawal Habibu O.[3], Lawal Balkis A.[4]**

*[1,2,4]Computer Science Department, Federal Polytechnic Ede, Nigeria*
*[3] Retail Operations Department, [9]Mobile Telecommunication Company, Ilesa, Nigeria*

Corresponding Authors: olawal70@gmail.com

**Abstract: Monolithic against microservices has always been a subject of debate in the developer world. Since developers now favor distributed application development approaches, microservies architecture has gained more favors over the last years. However, monolithic architecture is been used as an option in many other situations. The current organizational growth is moving monolithic to SOA and Microservices. Understanding the major differences between architectures can better benefit software development process. The right architectural choice depends on several factors, particularly when it comes to management capabilities and developer expertise. Microservices architecture employs small modules, self-sustaining which can be deployed and scaled separately. Microservices users believe that it represents the death of monolithic application software architecture. However, despite the benefits of microservices, the monolith architectural style is still standing its ground and still applicable in some area of software development. Adopting monolithic architecture for developing less complex application is usually the best option for small scale application development. This paper explores several advantages of monolith architecture over microservices and the other way round. Explaining the need to be sure what you need before starting your application development.**

**Keywords: Monolithic Architecture, Microservices Architecture, Service-Oriented Architecture, Distributed network, Enterprise Resource Planning, Enterprise Resource Planning,**

## 1.0 Introduction

The careful observation of the growth in the modern industry shows that autonomy and flexibility in production process is key factor in the development process of any organization. Companies agility and the ability to easily adjust to changes in the environment by quickly renovating its product and services, is becoming critical to retaining a competitive advantage over others organization in the same production line. A key factor in attaining such feet is to constantly monitor new innovations in Information and Communication Technology (ICT).

Many organizational capabilities and functions are underpinned by large monolithic software systems, including Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM); where monolith software has been defined as "a software application whose modules cannot be executed independently"[1]. The nature of software development is majorly based on monolithic architecture which is a big container that contains components of the software which are tightly packaged and bundle together, and each component is dependent on each other. As the companies that employ such system grow, so does the complexity of the software application been used. This inadvertently means that making changes to application become more difficult.

Industrial evolution has been trending from monolithic architecture towards Service-Oriented Architectures (SOA) and microservices (MA). Understanding the functionality, features and differences between these architectures will help build better microservice architecture and also help prevent the problems with the monolithic and SOA [2].

## 2.0 Monolithic Architecture

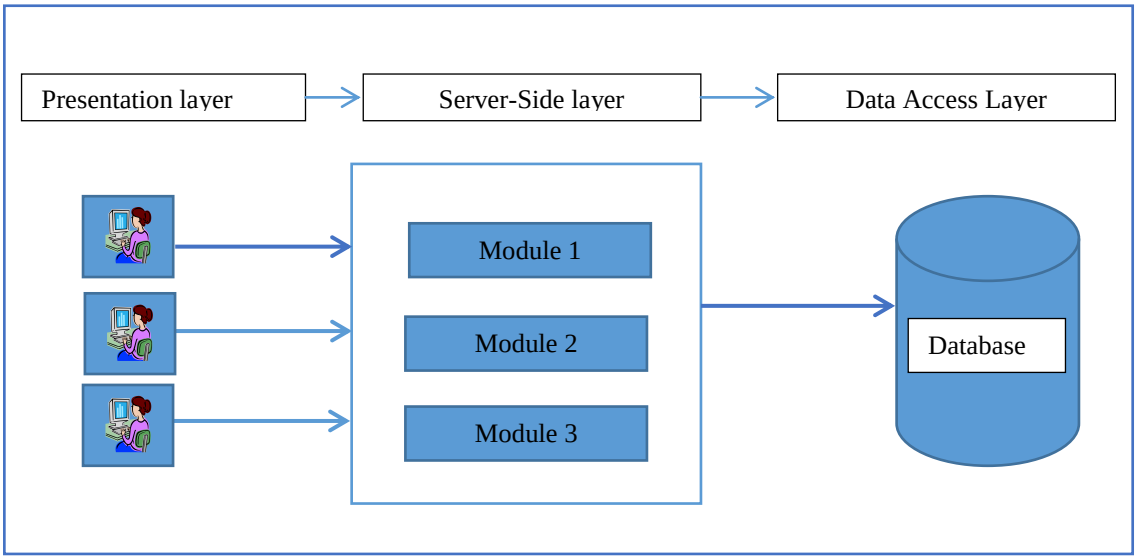
A monolithic architecture is a layered architecture in which each part of the architecture most interact with each other, the whole system is dependent on each other to perform its function. Monolithic architecture are usually designed to consist

a. Client side (presentation layer) – this is users interface which allows interactions with the system allowing data entry into the system.

b. Data Access layer – this is where data are stored for easy retrieval and usually allows CRUD functionality of the system.
c. Server-side layer – this part allows transactions between the user and Data Access later which is hidden from the Client-side layer.

In a monolithic architecture, all the functionalities have bound the services into one whole set of single code-based application. The whole application code is developed and deployed in a single project. This is due to the fact that enterprise-level applications need multiple developers, and whenever they want to update a single server resource they need to make sure that other services are not broken and will run smoothly. This adaption needs a lot of time and human resources to maintain constant functionality. Due to competition, businesses want to add new features to their application, which increases the nature of complexity in this type of approach and limits applying their innovative skills into the application. When a new update is attached to a service to make an application run into production, whole sets of servers need to be restarted. This impacts the user experience, which is not at all a good practice in this modern situation.

However, at some point, moving to microservices comes with some barter when microservices are poorly implemented or used as a patch without addressing some of the root problems in your system, the developer be unable to do new product development because the complexity of the application will be too difficult to comprehend [11].




**2.1 Problems with Monolithic Architecture**

Monolithic applications are usually very simple to design at the initial stage, but as the application grow and due to its tightly-packaged architecture which usually consist of the client-side layer, the data-access layer and the server-side layer, the architecture eventually as limitations and the risk factor also increases. In monolithic application technology, stack dependency stops the introduction of the latest technologies from the outside world. The present stack poses challenges as the web service itself suffers from some challenges such as the risk and limitations of monolithic architecture.

**2.2 Strengths of Monolithic Architecture**

a. **Overlapping concerns:** These are issues that affect the totality of the application. Monolithic architecture has to do with only one package, that is, the whole of the application itself and it usually much easier to handle.
b. **Debug and testing:** Compared to SOA and microservices architecture, since monolithic applications are a single packaged design, debugging is much easier and testing using an end-to-end method is much faster.

c. **Deployment:** Deploying monolithic applications are much easier to deploy. Since you do not have to handle multiple small modules.
d. **Development:** Using a monolithic approach as a standard for a team of developers, it much easier to develop applications if the team members have the right knowledges and understanding.

### 3.0 Microservice Architecture

Making software applications autonomous where each component within the application programming interface (such as API request, messaging component etc.) can be independently hosted within different deployment system is the major trending system in large organizations. This has brought about a decrease in software development using monolithic architecture. **Microservices architecture** breaks down application into a collection of smaller independent units. Each of the units within the application processes their services separately. Which means that each service as its own functional logic and a direct access to the database [8].

Application developments are becoming more autonomous, where each components of the application can be separately established using different deployment systems. Monolithic approaches have recently decline by building blocks of working components into separate deployment machines and allowing to link to the main module. This allows all the small blocks of applications to use loose coupling interfaces to communicate through either synchronous HTTP channels or use asynchronous messaging channels.

This application development approach is a migration towards microservices application development design method. Employment System development methods such as Agile and Scrum, a software development process can be broken down into multiple phased modules that can be designed and developed. Using this approach, all the applications can be integrated, deployed and phases testing can be done on each module. This design approach can be said to resemble microservices architecture development style.

*"The process of developing a software using blocks of self-sustaining applications with each block managing its own activities, liaising with light machines and using API resources is referred to as microservices architecture. These resources are designed with business potentials and can be deployed independently on a fully automated machine. There is a centralized management of all the processes and this can be designed using any programing language and database technology of the developer's choice" [10].*

Microservice architecture, also known as just "Microservices," is an approach to building software in which applications are separated into components called services, which are loosely coupled but function autonomously. Services are usually deployed independently so that a failure or outage of one does not affect the others. They are typically closely aligned with a particular business function or objective. The entire functionality is split up into independently deployable modules which communicate with each other through defined methods called APIs (Application Programming Interfaces). Each service covers its own scope and can be updated, deployed, and scaled independently.

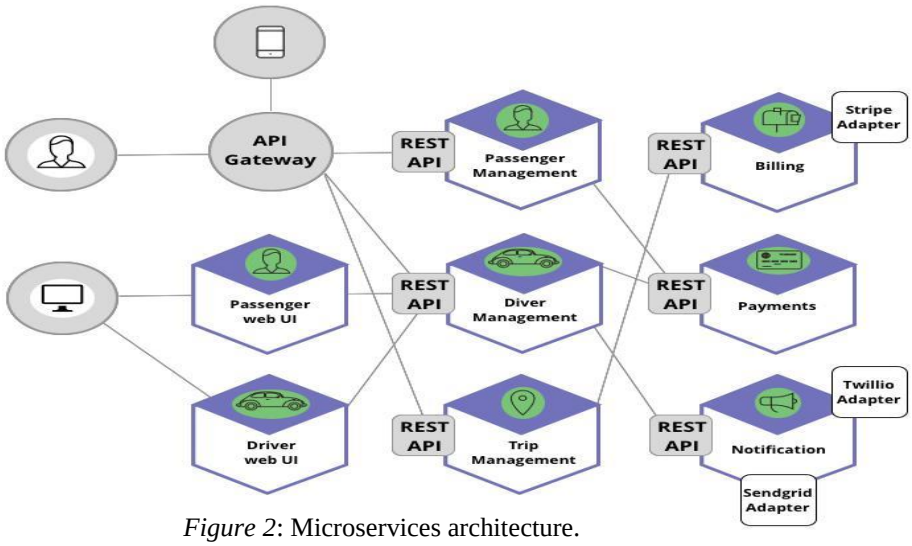



*Figure 2*: Microservices architecture.

### 3.1 Advantages of Microservice Architecture

a. **Self-standing Components:** Service deployment and updating can be done individually. An error in a block or module will not affect any other block within the entire application design. It is also not difficult to implement new designs in microservice architecture.
b. **Easier Understanding:** Reducing the design to small and easy to understand blocks, they are simpler, easy to understand and manage. All your efforts can be directed to the goal of achieving the related business need.
c. **Scalability:** Microservices approach allows independently scaling each block or modules. Scaling in microservices is much easy because each element in the application can be scaled independently. It is cost and time effective in the scaling of the entire process of microservices architecture. Unlike monolithic where the entire application has to be scaled where there is no need for it. Every monolithic application as limitations when it comes to scalability of applications, this therefore makes company ending up rebuilding the entire application. A very good example of microservices architecture is Netflix which follows the developmental process of microservices. This allows communications of blocked self-sustaining, battle-tested code as libraries that other developers can use to solve similar problems [10].
d. **Flexibility:** Developing an application using teams and choosing a particular technology as limited the choice of the development process from the start. Teams using microservices are allowed to choose their own programming languages.
e. **Agility:** Fault tolerance is usually applied in microservices architecture as faults in application affects a small part of entire process and not the entire application. There is low risk in experimentation and changes within the application and fewer bugs exist within the system.

## 4.0 Pitching microservice against monolithic

### 4.1 Network Latency

Since microservice involves making calling to services over a network, and this results in movement of bytes over the network. Hence, getting the latency of a microservice when it makes a call to a service is usually within or at least 24ms [13]. Therefore, if we imply that the definite processing of a call to a service takes about 100ms, then the total execution time can be show below;

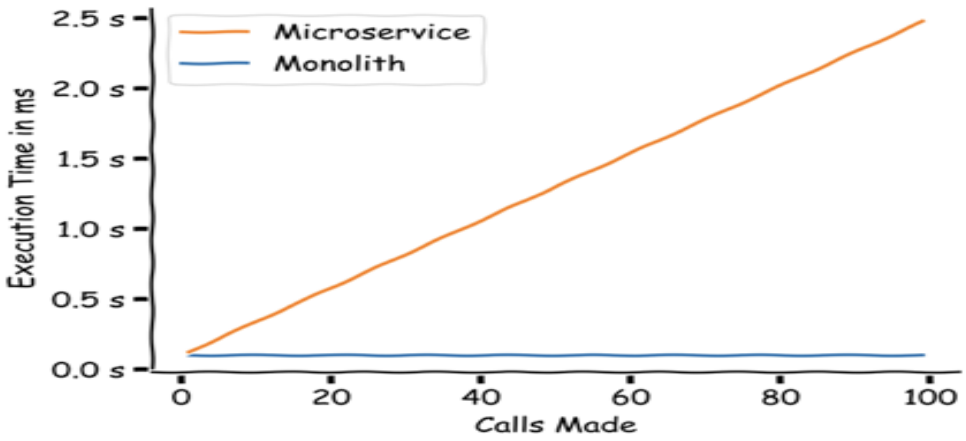



*Figure 3: Network latency – monolithic and microservice* (Romana, 2018).

The above execution is dependent on the fan-out pattern, which means that all execution is done at the same time and do not rely on each other. In figure 3, it shows that monolithic as no network latency because all calls are made locally within the compact module. And the way to fix this in a microservice is to reduce the distance of the calls to the needed services using fan-out and make data closely and local to the services.

### 4.2 Reliability

Since the execution of monolithic is all local, the chance of failure within the network is almost impossible. However, when microservice makes call from one service to another, the reliability drop by 0.1%. This means that when a microservice makes 1000 calls within a network, there is a possibility of one failing due to a network problem. And as the chain of calls within the services grow the reliability of the system drops – this means that out of every 100 call 1 will fail due to the growth in the chain of the calls [13].

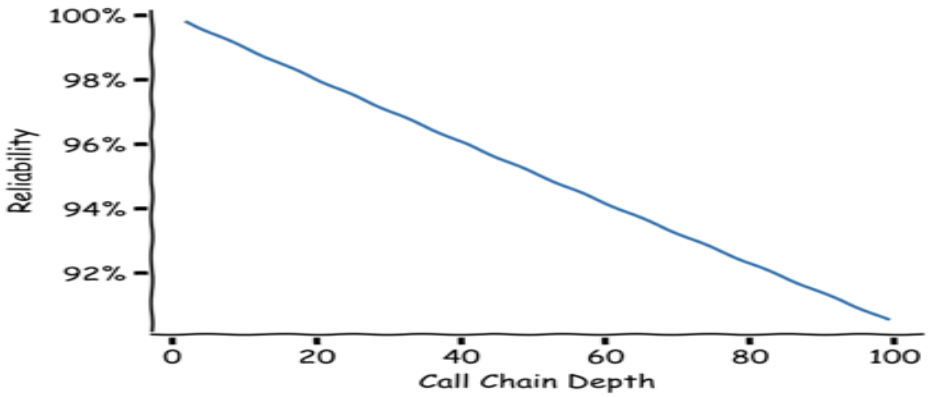



Figure 4: Reliability according to calls (Romana, 2018).

Figure 3, shows that the reliability of microservice reduces as the depth of the calls increases. It important to put into consideration the problems that comes with network failure when designing microservice architecture.

### 4.3 Complexity

When dealing with application development, there are a number complex issues to deal contend with;

a. Development complexity: This refers to the size of the application that has been designed. The code base can easily grow as the application development takes shape using microservices architecture. Microservices are suite of small services, which involves multiple source codes, duplication of codes and using different programming languages. Also, divergent process may use divergent versions of libraries, as release timetables are not synchronized.
b. Running complexity: Running and monitoring applications can be very complex, because microservices as to serve multiple modules, the amount of affected services is highly relevant. While a monolith architecture communicates with itself. A single call in microservice architecture can hit multiple services.

### 4.4 Resource Usage

Microservice uses more resources than a monolithic architecture. However, microservice uses resources smarter than monolithic. Microservices use cluster manager to allocate resources as needed and this results in low usage of resources. On the other hand, monolithic uses 20% of its codebase out of the entire 100% code. For example, if a process of a monolith uses 8GB, two processes with use combined 16GB resources. If 20% of codebase is executable, they will be executed in parallel to do the heavy lifting of the entire application. If a process of microservices runs on a 8GB, using 20% of the RAM which is 1.6GB, it only means that, if two process is running we will have 9.6GB of RAM usage. The diagram below shows the difference in resource usage [12][13].

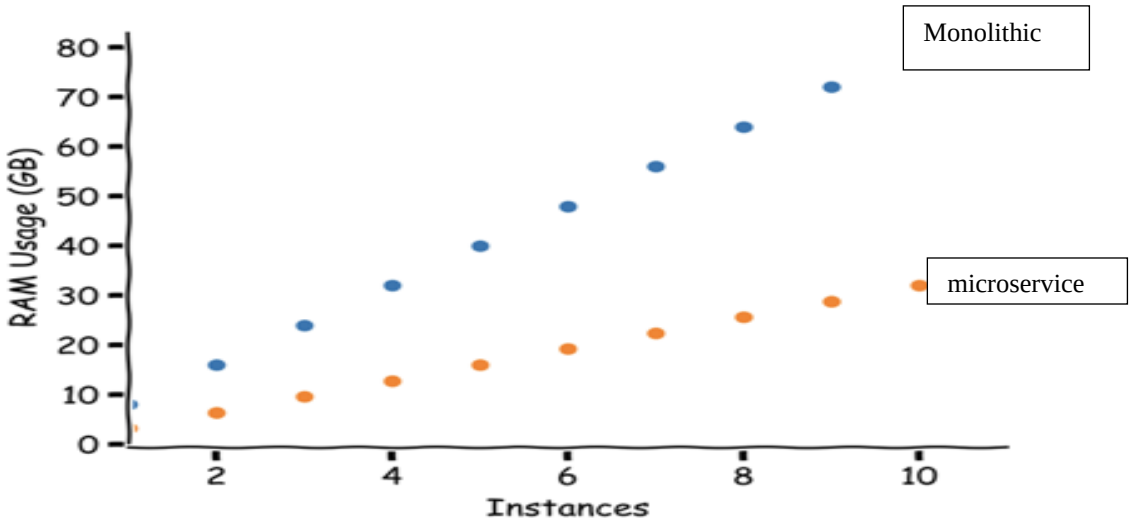



*Figure 5: Resource usage (monolithic and microservices,* Romana, 2018)

**4.5 Scalability**

Scaling a monolithic architecture is more difficult, it requires more resource usage and money. Scaling a monolithic requires that you run multiple instances of the same module in other to avoid downtime and lose of connection. Microservices deliver more throughput and more prices scaling is possible since you do not have a scale the entire application.

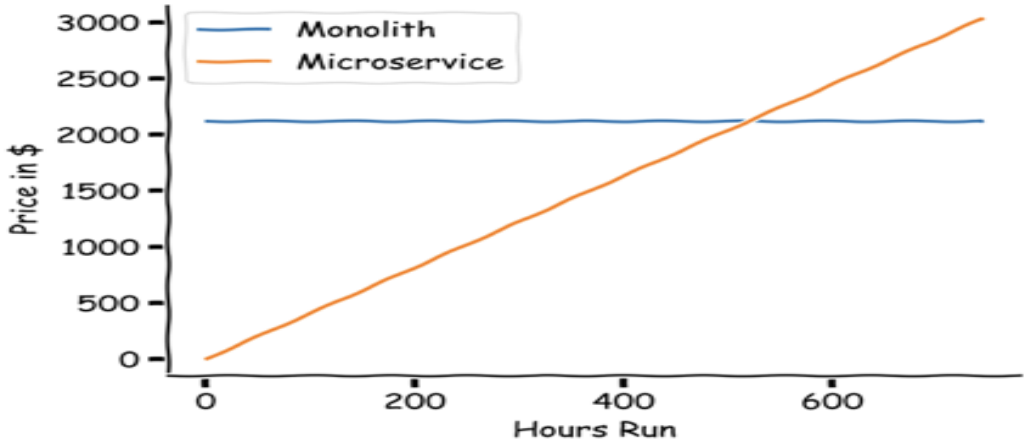



Figure 6: hours running compared to cost (Romana, 2018).

The figure 6 above shows running instances of microservices and monolithic using hours and the cost of running the instances.

**4.6 Time to Market**

Because monolithic architecture is a tightly packaged application, which contains its dependencies within itself, they are typically harder to deploy. However, deploying microservices are much easier to do. The developer can change a part of the main core system without modifying any of the dependent services. Versioning are clear and any changes usually as no clear impact on the overall application. Monolithic are usually object oriented, and it requires very dedicated team of programmers to make changes and not affect the core system. Microservices are also easier to test, have limited set of features and their dependencies are much more less. And because they are less intensive and are most build to scale, rolling out new features takes less time. All

these shows that microservice is a better option in terms of reduce in time from inception to production and deployment.

### 4.7 Communication

Building a large scale microservice application usually involves a development team. For instance; Amazon uses two-pizza rule. The rule is that working team should be minimal enough to eat two box of pizza and the goal of the rule was not to reduce the cost of catering. The idea behind it is to limit number of communications within the team. Teams only need to communicate through service interfaces. With a team of two people, there is one communication channel. With four people that could go up to six channels. A talk to person B, C and D. B talks to C and D and C talks to D. The formula for the number of communication channels is N (N − 1) / 2.

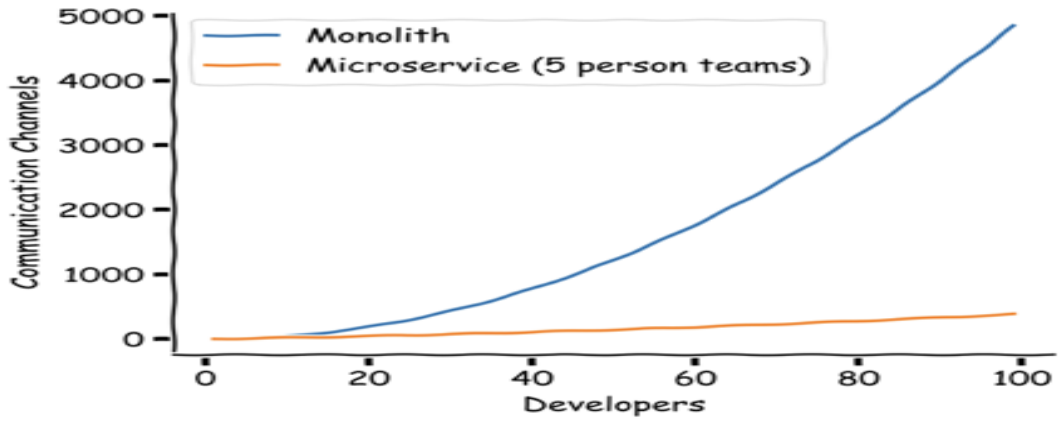



Figure 7: Team members against communication channels (Romana, 2018)

The figure 7 above shows that microservices model as better advantage over monolithic model. At the team with 20 developers, it shows that the number of communication channels is almost 10 times higher microservices than in the monolithic architecture. This allows work to be broken down in smaller teams and each team handling well defined small services.

### 5.0 Table 1: Comparison for Microservice Architecture and Monolithic Architecture

| Comparison | Monolithic | Microservices |
|---|---|---|
| **Simpler** | Monolithic has less moving parts, because it as a single code repository and a single log file. Easier to manage | More complex as the application grows |
| **More Reliable** | Execution of monolithic is all local, the chance of failure within the network is almost impossible | Allows easy distribution across multiple servers and if one server fails, other servers can take over. They are built on the assumption that networks are unreliable. |
| **Lower Latency** | Calls inside a monolithic are in process. All calls are made locally within the compact module | Calls are across the network. This adds overhead due to require sending and receiving a call over the network. |
| **Less Resources** | The have a limited scope, requires scaling every aspect of the project. | Allows scaling only the necessary component for a workload. |
| **Easier Scaling** | Services can be easily scaled up using less resource usage. | Not easy to scale |
| **Higher Throughput** | Have a lot of overhead and low throughput | They don't have overhead of image and orchestration. Where performance matter, monolithic is ahead in terms of |

| | | |
|---|---|---|
| | | throughput |
| **More Agile** | Due to efficient communication and small deployment footprint, they have minimal time to market. | Less agile due to constant redeployment after every single upgrade and as high time to market. |
| **Less Communication** | Communication channel increases as the team increases. However, by breaking down the team into smaller units, the channels can be greatly reduced | Fewer channels of communication exit in application development. |

**6.0 Monolithic architecture as An Option**

a. **Small Team:** Starting a project with a small team and building a very simple application does not require microservices architecture. The use of monolithic architecture will suffix for a small business need. There is no need to implement microservices due to its hyped users.
b. **A Simple Application:** Small businesses do require a lot business logic, scalability and the flexibility of implementation are also easy and they work better using monolithic architecture.
**c.** **No Microservices Expertise**: Building an application that require the use of microservices involves high end expertise. Designing an application using microservices architecture from the beginning without the required skills will probably end badly. While monolithic does not require learning and understanding multiple programming language and multiple technologies.
**d.** **Quick launch**: Designing and deploying an application as soon as it is completed is very easy with monolithic architecture. It works well when you aim to spend less initially and validate your business idea.

**6.1 Microservices Architecture As An Option**

**a.** **Microservices expertise:** Building an application using Microservices architecture without required expertise and understanding is very dangerous. Have the knowledge of the architecture is not a reason for adoption of the architecture. There is need for clear understanding of DevOps systems and Containers for the application development since the designed system are tightly coupled using microsevices.
**b.** **A complex and scalable application:** Scaling an application and addition of new design capabilities to an application is much easier using microservices architecture. If developing a very large application with various types of functionalities and module is what as application developer as in mind, using a microservices pattern is probably the best option for the developer.
c. **Enough engineering skills:** The use of multiple teams working on multiple services is usually employed in microservices. Hence, the need for large enough resources to handle all the processes.

**7.0 Conclusion**

This paper attempts to the shed more light on monolithic against microservices architecture, and their application areas. Emphasis is made on their growth and development process. Microservice architecture is a trending architecture being adopted by major large organization (Netflix, Amazon etc.) and it seems to be the more preferred option and software development in current time. However, application development of small and less complex application using monolithic architectures is usually the best option. Understanding that a move towards microservices with a team with little or no enough experience in distributed architectures is a very bad idea. One reasonable argument we've heard is that you shouldn't start with microservices architecture. Instead begin with a monolith, keep it modular, and split it into microservices once the monolith becomes a problem. Although this advice may not be the best ideal, since a good in-process interface is usually not a good service interface. However, using microservices architecture in developing complex system and employing developers familiar with microservices and motivated to achieve the application development is the best choice for the organization.

**REFERENCES**


1. Sasa Baskaradaa, Vivian Nguyenb, and Andy Koronios, Architecting Microservices: Practical Opportunities and Challenges 2018.
2. Keshab Katuwal, Microservices: A Flexible Architecture for the Digital Age Version 1.1 2016
3. https://subscription.packtpub.com/book/web_development/9781785887833/1/ch01lvl1sec9/understanding-problems-with-the-monolithic-architecture-style
4. Tomas Cerny, Micheal J. Donahoo, Michal Trnka, Contextual understanding of Microservice Architecture: Current and Future Direction. 2017.
5. Paolo Di Fancesco, Patricia Lago, Ivano M, Architecting with Microservices: A systematic Mapping Study
6. Joydip Kanjilal, (17 jan 2020), advantages of monolithic architecture that prove it isn't dead (online). Available: https://searchapparchitecture.techtarget.com/tip/Advantages-of-monolithic-architecture-that-prove-it-isnt-dead
7. Franchiti, J C (1992). Software Engineering G22.244-001. Courant Institute of Mathematical Science.
8. Romana Gnatyk, October 03, 2018, Microservices vs Monolith: which architecture is the best choice for your business? (online) Availaible: https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/
9. Richards, M. (2016). *Microservices vs. service-oriented architecture.* CA: O'Reilly Media, Inc.
10. James Lewis and Martin Fowler, (2014), Mocroservices. (online) Available: https://martinfowler.com/articles/microservices.html#footnote-fine-grained
11. Thomas Betts, (2020), To Microservices and Back Again - Why Segment Went Back to a Monolith.
12. Alexander Kainz, (2020), Microservices vs. Monoliths: An Operational Comparison
13. Marco Marcuccio, (2017), Network overhead in microservices-architecture. (online) Available: https://github.com/Marcuccio/Network-overhead-in-microservices-architecture
14. Alex Hern, (2018), The two-pizza rule and the secret of Amazon's success. (online) Available: https://www.theguardian.com/technology/2018/apr/24/the-two-pizza-rule-and-the-secret-of-amazons-success