LTI
Let's Solve

A Larsen & Toubro Group Company

Point of View

6 Factors to Consider before Adopting Microservices Architecture

Microservices is the most sought-after architecture today. Not only are Microservices Architects insistent crazy about changing every other system architecture to Microservices architectures, but even business stakeholders seem to be sold on the advantages they can reap by switching to the Microservices architecture.

Software architecture is regarded as ideal if it solves the intended problem and is flexible and scalable. Paul Clements, author of Software Architecture in Practice has said,

> **"The best software architecture knows what changes often and makes that easy"**

But, Microservices has, in fact, gone a step ahead and brought the element of speed and safety to the change in the architecture.. Considerably, the major theme around the Microservices architecture is the ease of change with speed and safety. Now, while the benefits of Microservices architecture are undeniable, it comes with its own set of challenges and requirements. And switching to any new architecture will incur cost and effort and so we need to really consider whether we have the right use case for a Microservices Architecture before we decide to adopt it. Below are a few points that can help in this consideration.

1 Your existing architecture already provides benefits of Microservices

Less adhesion and more cohesion are the two major characteristics of well-written code and Microservices adhere to these two characteristics at their core. Microservices are independent modules loosely coupled with each other and each module performs a particular task. Separation of concern is one of the first few rules taught in programming classes for writing better code.

Concept of "Separation of Concern" in software engineering enables the different teams to focus on different areas of the application and work on them independently. This eases the code maintenance and minimizes the impact of change on other modules. Microservices architecture exploits the concept of Separation of Concerns and extends it to the independent deployment This gives the luxury to development team to manage the different services independently and ease of testing and scaling. However, if we have achieved a higher level of cohesion and low adhesion in different modules, then we might need to explore the other benefits of Microservice architecture before adopting it.

2 Suitable for large applications

Microservices architecture is ideal for large applications. In the case of large monolith applications, code deployment effort is significant even for a small code change which can be one of the valid reasons for adopting the Microservice architecture. When the scope and boundaries of a monolith application tend to grow with time, we should consider the Microservice approach.

In today's fast-paced and continuously changing world of technology, a computer software is nothing but the reflection of business problems, and providing the feasible solution to address them. However, if our monolith application with its erstwhile architecture is already capable of incorporating the changes with ease and deploying the application does not take significant time and effort, then moving to a Microservices architecture for the sake of it, does not add value.

---

3 Reusability or Replaceability (SOA or Microservices)

Programmers around the world have always been taught about the concept of reusability. Probably, reusability can be regarded as one of the most important aspects of the good and efficient program. Service Oriented Architecture (SOA) is a design approach, where several individual services collaborate to provide some set of functionalities and emphasis is given to the reusability. SOA provides a service that can be used by one or more applications to cater to a functionality and at its core, SOA is a simple well-written logic that can be reused by more than one application or module. However, Microservices focus on the ease of replaceability. where they are used by only one module or application. They are goal-oriented, and just a separate piece of software that may not make a business case and may not provide any value if considered as a standalone service. So, we need to have a clear understanding of the requirement whether we need a reusable piece of code (in which case an SOA should be the preferred architecture), or we want our application to be modularized to a level, where standalone deployment and scalability saves cost and effort with Microservices.

4 Orchestrate multiple services in harmony

A musical orchestra playing a particular tune in symphony can be taken as a perfect analogy to understand the concept of microservices. In musical orchestra, multiple artists play a series of chords with their different instruments in perfect harmony to produce a sweet musical tune. Similarly, in microservices architecture, multiple services collaborate to perform a particular task. Each service is a separate entity and can be deployed and scaled separately, but it must be part of the orchestra to provide the desired business value. This need for the perfect harmonization to run the orchestra brings its own challenges and complexity. All the Microservices which are part of the Microservices architecture-based system need to be maintained separately, with different teams working on the different services without communication barriers or distinction in understanding of the business value the application needs to deliver. Therefore, the ease of communication, well-defined logical boundaries, and scope of a particular Microservice are the key ingredients to achieve the cohesiveness and efficiency in the landscape of Microservice architecture. Linguistic barrier - geographically distributed but disconnected teams working in different time zones and the difference in understanding of the business value to be delivered - can be the major impediments to develop an application using Microservices architecture. It is suggested that people working on one service should be co-located.

---

5 Team with right skill set

One of the main characteristics of this architecture is that the Microservices are developed and deployed independently. Deploying a monolith is a straightforward process but deploying a Microservice is a different task altogether due to the interdependence. If the deployment strategy is not correct, then having a Microservice architecture can be a big pain. Most-suited development and deployment strategy for the Microservice architecture is Continuous Development and Continuous Integration. This deployment feature calls the need for adoption of the mantra "You built it, you run it" given by Amazon. This strategy ensures that a piece of code written by a developer not only works independently in his module but also integrates well with the other required modules and serves the desired purpose. This approach reduces the effort in fixing the integration issues during the release process and saves significant amount of rework. It absolutely makes sense that to

adhere to "You built it, you run it" mantra developer should have basic skills of deploying the code along with his primary programming skills. Having a development team with no exposure to the deployment strategies can be a bottleneck because it will present an impediment in the process of continuous integration. So, it is necessary for the development teams to have some level of exposure in deployment strategies along with the regular development skills.

The core idea behind the Microservices architecture is to respond the change in faster and safer way and hence, the Microservices architecture is suitable for the applications addressing rapidly changing business requirements. Agile methodologies are the most-suited software development methodologies for the development of applications which address continuous changing business requirements. In such agile development teams, the development and testing of the application goes hand in hand, and this requires the developer to be equipped with a certain level of testing skills and a relatively higher level of functional knowledge of the application.

Having separate development, testing, and deployment teams for each Microservice is a relatively costly affair and create unwanted dependency within the team. So, having a cross-functional team and full stack team members is the right preparation for adopting the Microservice architecture for efficient and cost-effective development.

---

6 Team willingness

Microservices architecture requires a paradigm shift in the behavioral aspect too. Agile mindset and better communication within the team are the catalysts for the smooth development of the application employing Microservice architecture. Management and business stakeholders too need to be prepared to facilitate the ease of communication within and between the teams and willingness to remove all the possible communication barriers for the smooth functioning of the teams. Open culture, lesser hierarchies and closely connected business and development teams act as the perfect catalyst for the efficient development process.

Conclusion

Before embracing the Microservices architecture, these six factors need to be considered while carefully weighing the pros and cons in the context with one's existing system architecture to analyze whether or not the Microservices architecture can resolve their problems. Microservices offer loads of benefits which are but subjective to the characteristics of the IT landscapes. Hence, adopting Microservice architecture should be a far-sighted and well-thought decision. Microservices architecture is not a one-stop solution for all the problems, but it can offer innumerable benefits if implemented correctly.

About the Author

Nilesh Dubey
Senior Specialist - Consulting, LTI

Nilesh is Project Manager for AI/ML platform Deployment & Support. He has 15 years of experience with expertise in Java/J2EE, Service Oriented Architecture (SOA) and Microservice Architecture solutions. He is an agile practitioner with experience on methodologies such as Scrum, Kanban, Extreme Programming (XP) using Test Driven Development (TDD), pair programming practices and SAFe framework. He is a Certified Scrum Master.