CHALLENGES OF MICROSERVICES & WHEN TO AVOID THEM



The Microservices Architecture—a variant of the Service-Oriented Architecture (SOA)—is an evolved development approach that has emerged from the world of domain-driven design that:

- Enables efficient computing
- Aids in building resilient applications

However, as with any approach to application development, the microservices architecture has its own challenges. There are also use cases where organizations notice increased complexity and effort to develop applications with a microservice framework.

This article will discuss such these challenges, including when you should not use microservices. Let's take a look.

Microservices overview

A microservices approach means designing and developing an application as a group of loosely coupled services that communicate among themselves to achieve a stated business objective.

A rising pattern of wider adoption rate confirms that there are several use cases in which businesses benefit from using microservices to develop and deploy applications. After all, microservices are often considered the first step to embracing a DevOps culture, which enables:

- Automation

- Improved scalability
- Manageability
- Agility
- Faster delivery & deployment

For the same reason, a microservice architecture is usually the first choice for businesses looking to rewrite legacy applications to use modern programming languages and technology stack.

For all these benefits, you might wonder what the challenges could be.

Challenges of microservices architectures

Now, let's turn to the most common challenges from companies or product teams adopt microservices.

Design

Compared to monolithic apps, organizations face increased complexity when designing microservices. Using microservices for the first time, you might struggle to determine:

- Each microservice's size
- Optimal boundaries and connection points between each microservice
- The framework to integrate services

Designing microservices requires creating them within a bounded context. Therefore, each microservice should clarify, encapsulate, and define a specific responsibility.

To do this for each responsibility/function, developers usually use a data-centric view when modeling a domain. This approach raises its own challenge—without logic, the data is nonsensical.

Security

Microservices are often deployed across multi-cloud environments, resulting in increased risk and loss of control and visibility of application components—resulting in additional vulnerable points. Compounding the challenge, each microservice communicates with others via various infrastructure layers, making it even harder to test for these vulnerabilities.



Data security within a microservices-based framework is also a prominent concern. As such, data

within such a framework remains distributed, making it a tricky exercise to maintain the confidentiality, integrity, and privacy of user data.

Due to its distributed framework, setting up access controls and administering secured authentication to individual services poses not only a technical challenge but also increases the attack surface substantially.

Testing

The testing phase of any software development lifecycle (SDLC) is increasingly complex for microservices-based applications. Given the standalone nature of each microservice, you have to test individual services independently.

Exacerbating this complexity, development teams also have to factor in integrating services and their interdependencies in test plans.

Increased operational complexity

Each microservice's team is usually tasked with deciding the technology to use and manage it. As each service should be deployed and operated independently, maintaining operations may open a can of worms for those who are not prepared.

Here are some challenges:

1. **Traditional forms of monitoring may not work for a microservices-based application.** Consider a scenario where a request from the user interface traverses multiple services before getting to the one that can fulfill its request. The result of this traversal is a convoluted path of services, and without the appropriate monitoring tools, identifying the underlying cause of an issue is not only tricky—it's often impossible.
2. **Scalability is another operational challenge associated with microservices architecture.** Although the scalability of microservices is often touted as an advantage, successfully scaling your microservice-based applications is challenging.
3. **Optimizing and scaling require more complex coordination.** In a typical microservices framework, an application is broken down to smaller-independent services that are hosted and deployed across separate servers. This architecture requires coordinating individual components, which is another challenge particularly when you experience a sudden spike in application usage.
4. **Fault tolerance needed for every service.** Businesses need their microservices to be resilient enough to withstand internal and external failures. In a microservices-based application, one component failing can affect the entire system. Therefore, the framework you use should consider fault tolerance for every service to ensure a design that prevents failure of an entire application in the event of an individual service downtime.

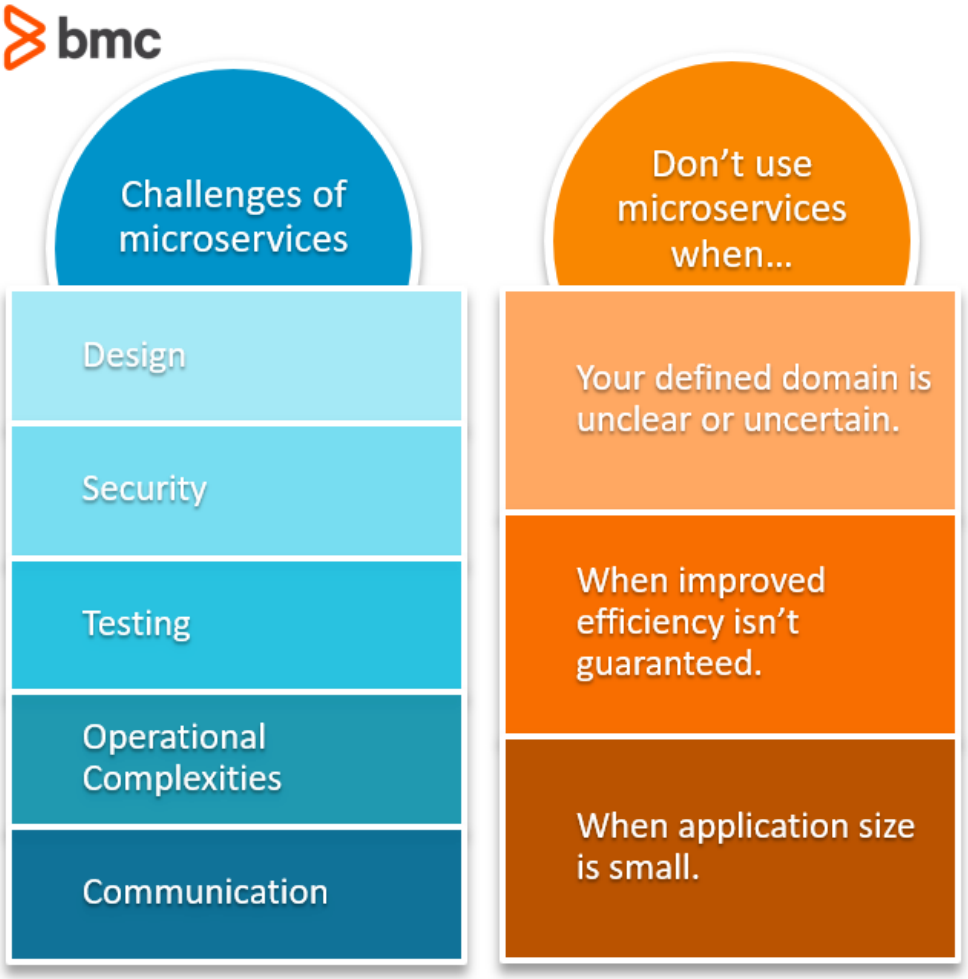
Communication

Independently deployed microservices act as miniature standalone applications that communicate with each other. To achieve this, you have to configure infrastructure layers that enable resource sharing across services.

A poor configuration may lead to:

- Increased latency
- Reduced speed of calls across different services

In this situation, you've got a non-optimized application with a slow response time.



When not to use microservices

Every organization shifting to a new framework should perform thorough due diligence to ensure it's the right fit.

When exploring microservices, these three situations will always help you decide when to skip the microservice architecture for your chosen application.

Your defined domain is unclear/uncertain

Recall that you create microservices within a bounded context. Therefore, if it is logically complex to break down your business requirements into specific domains, it will be equally difficult for you to create adequately sized microservices.

Add to that the challenge of designing a proper means of communication among different services—this complexity is likely too much for you to realize maximum benefits in microservices.

Also, consider the future. If you're not certain that your application's domain will remain the same over the coming years, it's wise not to use a microservices-oriented approach.

Improved efficiency isn't guaranteed

To reiterate, the idea of adopting Microservices is to embrace a DevOps culture that in turn:

- Employs automation
- Reduces cost and effort
- Brings operational efficiency

Carry out your due diligence to verify if transitioning to a microservices framework actually helps achieve these goals. No organization would like to add up complexities and effort just to adopt a culture without gaining improved efficiency.

Application size is small or uncomplex

When your application size does not justify the need to split it into many smaller components, using a microservices framework may not be ideal. There's no need to further break down applications that are already small enough as-is.

Remember that the idea of using a microservices framework is to break a complex application into a bunch of different, smaller services. When an application code is already small enough and straightforward—not complex—transitioning it to a microservice framework will only add complexities.

Are microservices right for me?

While adopting a Microservice Architecture offers faster delivery and improved software quality, businesses should deliberate carefully whether to opt for this approach. The benefits of transitioning from a monolithic to microservices model are a dime a dozen—but there are still challenges to consider.

The general rule of thumb for choosing microservices:

If the cons of using microservices outweigh the benefits, and/or the benefits are negligible compared to the effort and money spent, microservices likely are not the right approach for the application.

Related reading

- BMC DevOps Blog
- The Role of Microservices in DevOps, part of our DevOps Guide
- 15 Best Practices for Building a Microservices Architecture
- What is SOA? Service-Oriented Architecture Explained
- What is Behavior-Driven Development (BDD)?
- Docker Monitoring: How to Monitor Containers & Microservices