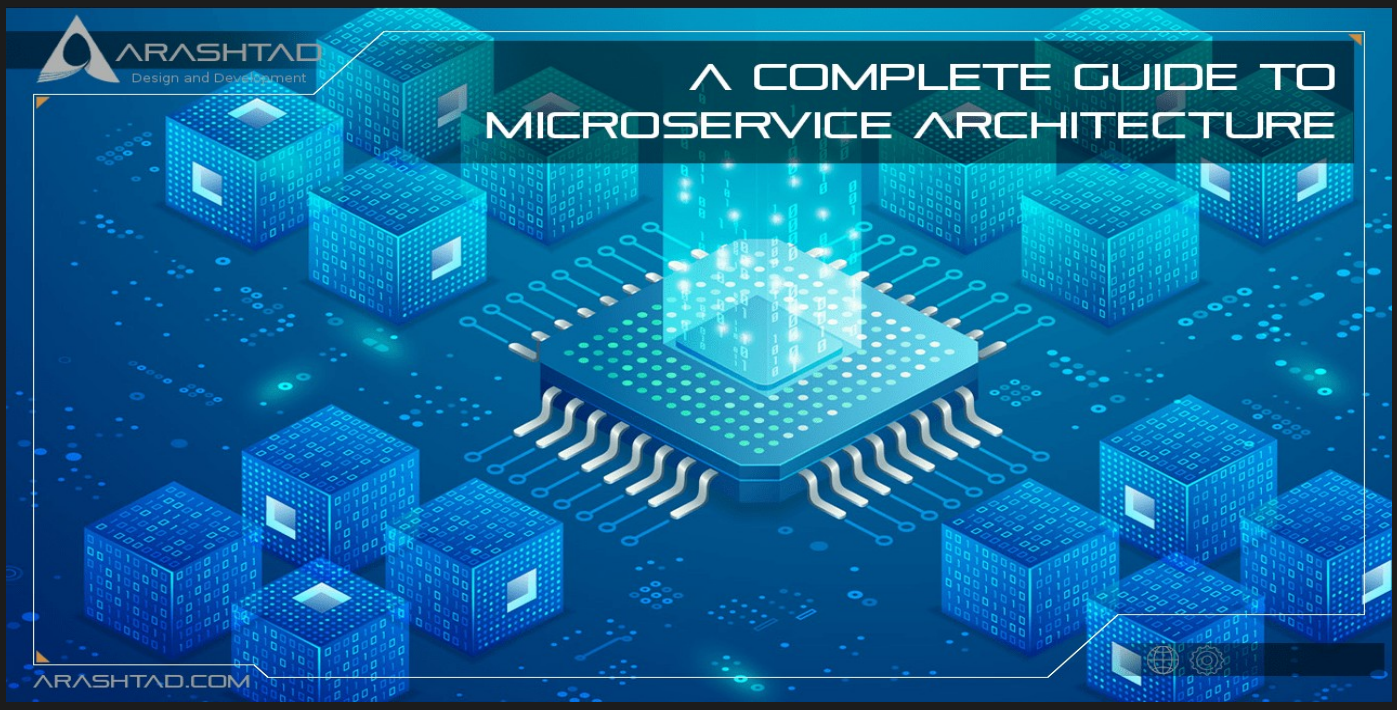
Title	A COMPLETE GUIDE TO MICROSERVICE ARCHITECTURE
Description	Intro
Date	2022 14 Augus
Author	Arashtad
Author URI	https://Arashtad.com



An architecture based on microservices consists of a series of small, autonomous services. Each service is self-contained and should implement a single business capability within a bounded context. a bound context is a natural division within a business and provides a definite boundary within which a domain model can be applied. In a microservice architecture, services are loosely coupled and can be developed, deployed, and maintained independently. These services each handle a discrete task and can communicate with other services using simple APIs to solve larger complex business problems. In this article, we will take a close look at the different aspects of microservices, what they are, What are the challenges and benefits of using them and so on.

WHAT ARE MICROSERVICES?

A microservice is a small, independent, and loosely coupled service. It can be written and maintained by just one or two developers. Each service has a separate codebase, which can be managed by a small team. There is no need to rebuild and redeploy the entire application when updating an existing service. Services are responsible for persisting their own data.Data persistence is handled by a separate data layer in contrast to the traditional model. services communicate with one another via well-defined APIs. internal implementation details of each service are hidden from other services. polyglot programming is supported. Services do not need to share the same technology stacks, libraries, or frameworks, for example.

THE PROS AND CONS OF MICROSERVICES:

It is possible to build the constituent services by one or more small teams from the beginning separated by service boundaries, making it easier to scale up development efforts in the future. Once developed, these services can also be deployed independently of each other, making it easy to identify hot services and scale them independently from the whole application.

Another benefit of microservices is also improved fault isolation, which means that the whole application does not necessarily stop working in the event of an error in one service. if the error is fixed, a smaller version of the application could be deployed instead of a whole app being re-deployed. One of the advantages of microservices architecture is that you can choose which technology stack (programming languages, databases, etc.) is best suited to the required functionality (service) instead of having to use a more standardized, one-size-fits-all approach.

Pros:
Easier Debugging:
Managing bug fixes and feature releases is easier with microservices because they are deployed independently. you can update a service without redeploying the entire application, and you can roll back an update if something goes wrong. When a bug is discovered in a traditional application, it can stall the entire release process. new features may be delayed while a bug fix is integrated, tested, and published.
Smaller teams are needed:
It is important to use microservices that are small enough to be built, tested, and deployed by a single team. Small teams are more agile than large teams because of slower communication, increased management overhead, and diminished agility.
Small code base:
It is easy for monolithic applications to become tangled over time due to a high number of code dependencies. adding new features requires touching a lot of code. Adding new features is easier with a microservices architecture because it does not share code or data stores.
Scalability:
It is possible to scale out services independently, allowing you to scale out subsystems requiring more resources without scaling out the entire application. with orchestrators like Kubernetes and Service Fabric, you can load more services onto a single host, which allows for better resource utilization.
Data isolation:
The process of performing schema updates is much simpler, because only one microservice is affected. Schemas updates can be challenging in a monolithic application because different components of the application may all interact with the same information, making any changes to it risky.
Variety of Options:
Teams can choose any technology that fits the need of their service. For instance they can choose MySQL or MongoDB for the database, with Django and Python, Docker, Redis and so on. They have the option of choosing the framework, language, database and the kind of tools necessary for their service.
Fault Isolation:
Microservices can become unavailable, but the entire application won’t go down as long as any upstream microservices can handle faults correctly (for example, by implementing circuit breaking).
Cons:
With every benefit comes a challenge and every opportunity creates a threat. We have the same story for microservices.
Testing and Development:
It requires a different approach to writing a small service that relies on other dependent services than it takes to write a traditional monolithic or layered application. Existing tools are not always designed to deal with service dependencies. Refactoring across service boundaries can be challenging. Testing service dependencies can also be challenging, especially when the application is rapidly evolving.enough to handle microservices.

Issues of Decentrlization:
Decentralized microservices have many advantages, but they can also cause problems. the application may become difficult to maintain if you use so many different languages and frameworks. Standardizing project-wide functionality without overly restricting teams' flexibility may be helpful. this is especially true for cross-cutting functions like logging.
Network congestion and latency:
There will be more interservice communication if there are many small, granular services. Furthermore, if the chain of service dependencies becomes too long (service A calls service B, which then calls service C.), the increased latency can become a problem. The design of APIs must be carefully considered. avoid overly chatty APIs, consider serialization formats, and find ways to use asynchronous communication methods like queue-based load levels.
Data integrity:
Due to each microservice being responsible for its own data persistence, data consistency can be a challenge. Embrace eventual consistency when possible.
Management:
In addition to mature DevOps practices, microservices require correlated logging across services. To log a single user operation, multiple service calls must be correlated.
Versioning:
It is possible for multiple services to be updated at any given time, so if you don't carefully design things, you may experience problems with forward compatibility and/or backward compatibility.
Various Skill Set:
Since any microservice require various talents, it is important to determine whether the team is skilled and experienced enough to handle microservices.

MICROSERVICE USE CASES:

A microservice architecture is built using Java, especially Spring Boot, to speed up application development. microservice architectures are often compared with service-oriented architectures. they both have the same objective, which is to separate monolithic applications into smaller components, but they have different approaches. Here are some microservices architecture examples:

Website migration:
It is possible to migrate a monolithic website to a microservices platform based on cloud computing and containers.
Media content:
Object storage systems offer scalable storage for images and videos, and they can be served directly to web or mobile devices using a microservices architecture.
Transactions and invoices:
Despite not being able to process payments, orders can be processed independently, so payments can continue to be accepted even if invoicing does not work.
Data processing:
Existing modular data processing services can be extended to the cloud with the help of a microservices platform.

CONCLUSION

In this article, you learned about Microservices what they, their architecture, design, implementation, benefits, challenges, and use cases. Microservices are so common these days and companies and startups are hiring developers with different expertise who can handle a part of the microservice development. These are of course various parts with their own specific skill set that requires a team of developers.