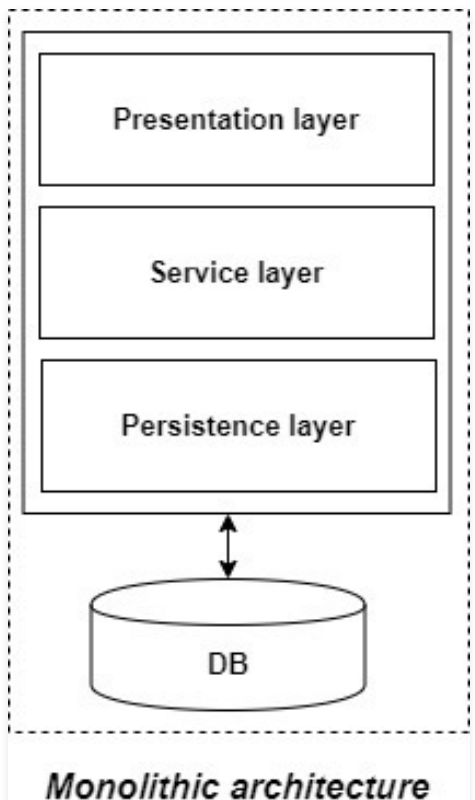


In order to understand microservices, we need to understand what are monolithic applications and what led us to move from monolithic applications to microservices in recent times.

Monolithic applications

If all the functionalities of a project exists in a single codebase, then that application is known as monolithic application. We all must have designed a monolithic application in our lives in which we were given a problem statement and were asked to design a system with various functionalities. We design our application in various layers like presentation, service and persistence and then deploy that codebase as single jar/war file. This is nothing but a monolithic application where “mono” represents the single codebase containing all the required functionalities.



But if we were using monolithic applications already then what led us to microservices?

Disadvantages of Monolithic applications:

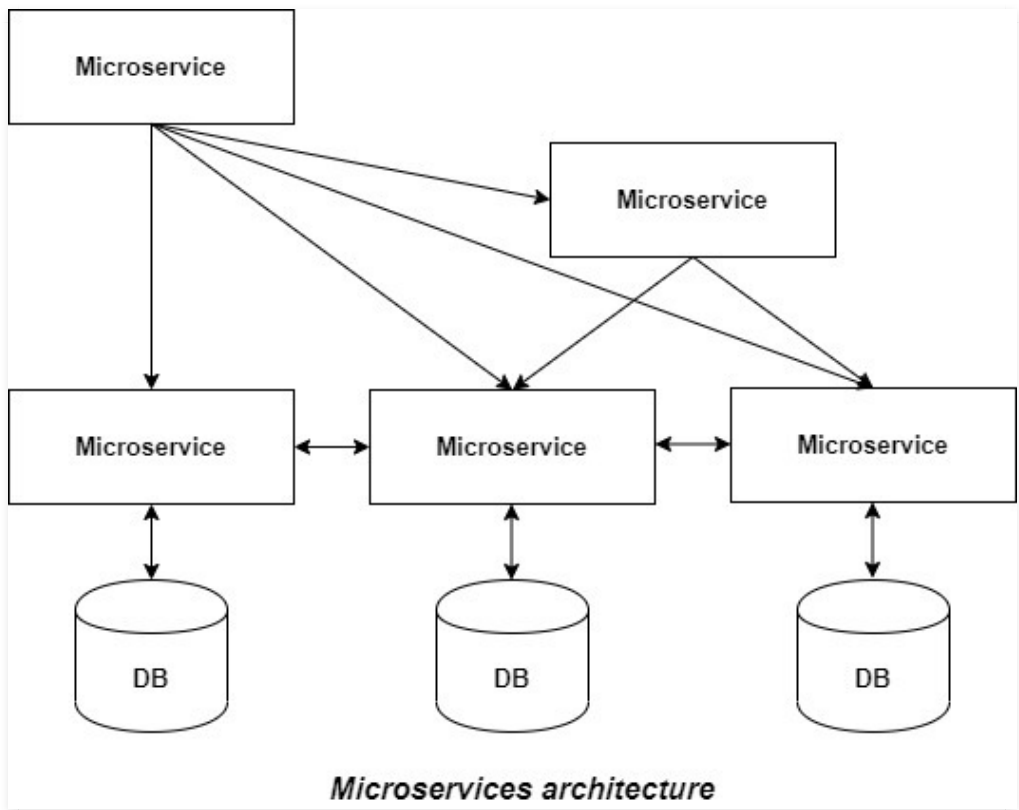
- It becomes too large in size with time and hence, difficult to manage.
- We need to redeploy the whole application even for a small change.
- As the size of the application increases, its start-up and deployment time also increases.
- For any new developer joining the project, it is very difficult to understand the logic of large Monolithic application even if his responsibility is related to a single functionality.
- Even if a single part of the application is facing a large load/traffic, we need to deploy the instances of the whole application in multiple servers. It is very inefficient and takes up more resources unnecessarily. Hence, horizontal scaling is not feasible in monolithic applications.
- It is very difficult to adopt any new technology which is well suited for a particular functionality as it affects the whole application, both in terms of time and cost.
- It is not very reliable as a single bug in any module can bring down the whole monolithic application.

Advantages of monolithic applications:

- Simple to develop relative to microservices where skilled developers are required in order to identify and develop the services.
- Easier to deploy as only a single jar/war file is deployed.
- Relatively easier and simple to develop in comparison to microservices architecture.
- The problems of network latency and security are relatively less in comparison to microservices architecture.

Microservices

It is an architectural development style in which the application is made up of smaller services communicating with each other directly using light weight protocols like HTTP. According to Sam Newman, “Microservices are the small services that work together.”



The Microservice architecture has a significant impact on the relationship between the application and the database. Instead of sharing a single database with other microservices, each microservice has its own database. It often results in duplication of some data but having a database per microservice is essential if you want to benefit from this architecture as it ensures loose coupling. Another advantage of having separate database per microservice is that each microservice can use the type of database best suited for its needs.

Principles of microservices:

- **Single responsibility:** It is one of the principles defined as a part of SOLID design pattern. It states that a single unit, either a class, a method, or a microservice should have one and only one responsibility. Each microservice must have a single responsibility and provide a single functionality. You can also say that: the number of microservices you should develop is equal to the number of functionalities you require. The database is also decentralized and generally, each microservice has its own database.
- **Built around business capabilities:** In today’s world, where so many technologies exist, there is always a technology which is best suited for implementing a particular functionality. But in monolithic applications, it was a major drawback, as we can’t use different technology for each functionality and hence, need to compromise in particular areas. A microservice shall never restrict itself from adopting appropriate technology stack or backend database storage which is most suitable for solving the business purpose i.e. each microservice can use different technology based on business requirements.
- **Design for failure:** Microservices must be designed with failure cases in mind. Microservices must exploit the advantage of this architecture and going down of one microservice should not affect the whole system, other functionalities must remain accessible to the user. But this was not the case in the Monolithic applications, where failure of one module leads to downfall of the whole application.

Service oriented architecture (SOA) vs Microservices architecture:

Steve Jones, MDM at Capgemini once said, “Microservices is SOA, for those who know what SOA is”. So, those who know about SOA, mostly think that they are same or the difference is not much clear in their mind. We can’t blame them also, if we talk about a cake and a pastry, we will find more similarities than differences. So let’s try to understand the differences between the two.

SOA evolved in order to deal with the problems in the monolithic architecture and became popular in the early 2000s. In SOA, the large application is split up into multiple smaller services that are deployed independently. These services doesn’t communicate with each other directly. There used to be an Enterprise Service Bus (ESB, a middleware or server with the help of services using different protocols or message standards can communicate with each other easily) where these services expose themselves and communicate with each other through it. Also, there was no guideline to have independent database for each service.

Microservices architecture is an evolution of SOA. People also consider SOA as a superset of microservices. In simple terms, microservices is fine-grained SOA. Here, the microservices communicates with each other directly and there is no central dependency for communication such as ESB in SOA. Also, there is a guideline to have a separate database for each microservice. The fundamental idea of evolution of microservices from SOA is to reduce the dependency between the services and make them loosely coupled by above mentioned guidelines.

Advantages of microservices:

- It is easy to manage as it is relatively smaller in size.
- If there’s any update in one of the microservices, then we need to redeploy only that microservice.
- Microservices are self-contained and hence, deployed independently. Their start-up and deployment time are relatively less.
- It is very easy for a new developer to on-board the project as he needs to understand only a particular microservice providing the functionality he will be working on and not the whole system.
- If a particular microservice is facing a large load because of the users using that functionality in excess then we need to scale out that microservice only. Hence, microservices architecture supports horizontal scaling.
- Each microservice can use different technology based on the business requirements.
- If a particular microservice goes down due to some bug, then it doesn’t affect other microservices and the whole system remains intact, continues providing other functionalities to the users.

Disadvantages of microservices:

- Being a distributed system, it is much more complex than the monolithic applications. Its complexity increases with the increase in number of microservices.
- Skilled developers are required to work with microservices architecture which can identify the microservices and manage their inter-communications.
- Independent deployment of microservices is complicated.
- Microservices are costly in terms of network usage as they need to interact with each other and all these remote calls results into network latency.
- Microservices are less secure relative to monolithic applications due to the inter-services communication over the network.
- Debugging is difficult as the control flows over many microservices and to point out why and where exactly the error occurred is a difficult task.