Monolithic vs Microservices architecture

Monolithic applications
If all the functionalities of a project exist in a single codebase, then that application is known as a 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 a 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 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 a 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 entire 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 entire application, both in terms of time and cost.
- It is not very reliable, as a single bug in any module can bring down the entire 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.
- Developers need not learn different applications, they can keep their focus on one application.

Microservices

It is an architectural development style in which the application is made up of smaller services that handle a small portion of the functionality and data by communicating with each other directly using lightweight 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 a separate database per microservice is that each microservice can use the type of database best suited for its needs. Each service offers a secure module boundary so that different services can be written in different programming languages. There are many patterns involved in microservice architecture like service discovery & registry, caching, API gateway & communication, observability, security, etc.

Principles of microservices:

- **Single responsibility:** It is one of the principles defined as a part of the 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 that 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 an appropriate technology stack or backend database storage that 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 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 the failure of one module leads to the 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, most think that they are the same, or the difference is not much clearer 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 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 don’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 that 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 an 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 a fine-grained SOA. Here, the microservices communicate 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 the evolution of microservices from SOA is to reduce the dependency between the services and make them loosely coupled with the above-mentioned guidelines.

Advantages of microservices:

- It is easy to manage as it is relatively smaller.
- 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 times are relatively less.
- It is very easy for a new developer to onboard 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, the 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 and continues providing other functionalities to the users.

Disadvantages of microservices:

- Being a distributed system, it is much more complex than monolithic applications. Its complexity increases with the increase in a 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 result in 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.

SOAP v/s REST API

What is SOAP?

SOAP is a protocol which was designed before REST and came into the picture. The main idea behind designing SOAP was to ensure that programs built on different platforms and programming languages could exchange data in an easy manner. SOAP stands for Simple Object Access Protocol.

What is REST?

REST was designed specifically for working with components such as media components, files, or even objects on a particular hardware device. Any web service that is defined on the principles of REST can be called a RestFul web service. A Restful service would use the normal HTTP verbs of GET, POST, PUT and DELETE for working with the required components. REST stands for Representational State Transfer.

SOAP uses only XML for exchanging information in its message format whereas REST is not restricted to XML and its the choice of implementer which Media-Type to use like XML, JSON, Plain-text. Moreover, REST can use SOAP protocol but SOAP cannot use REST.

- On behalf of services interfaces to business logic, SOAP uses @WebService whereas REST instead of using interfaces uses URI like @Path.
- SOAP is difficult to implement and it requires more bandwidth whereas REST is easy to implement and requires less bandwidth such as smartphones.
- Benefits of SOAP over REST as SOAP has ACID compliance transaction. Some of the applications require transaction ability which is accepted by SOAP whereas REST lacks in it.
- On the basis of Security, SOAP has SSL(**S**ecure **S**ocket **L**ayer) and WS-security whereas REST has SSL and HTTPS. In the case of Bank Account Password, Card Number, etc. SOAP is preferred over REST. The security issue is all about your application requirement, you have to build security on your own. It’s about what type of protocol you use.
- SOAP cannot make use of REST since SOAP is a protocol without any architectural pattern. REST can make use of SOAP because it is an architectural pattern having protocol.