What is microservice ?

1) It is an architecture style which says decompose big application into smaller services later communicate these services together to form larger business application.

2) It is autonomous, self-contained and independently deployable.

3)It is not a programming language or framework.

4)There are many architectures like MainFrame architecture,client server architecture,MVC architecture,SOA architecture (service oriented architecture) and finally microservices architecture.So this is latest architecture.Who will use it architect. So architect will use microservice architecture to design the project.

5)This architecture says decompose means split project into smaller smaller services.

Example of Microservice:-

I have a brownfield airlines project and the project name is PSS project.

Previously the brownfield airlines project was developed in SOA later it was migrated to microservices architecture.

The microservices are

1)Fare microservice

2)Booking microservice

3)Search microservice

4)Check-in microservice

we are using RabbitMQ as messaging listener.

Different messaging listener are RabbitMQ/ActiveMQ/Kafka

Different HTTP listener are :-tomcat ,jetty,JBoss,weblogic server etc.

Http listener are used for syncronous communication.

Messaging listener are used for asyncronous communication.

Microservices are Self contained means every microservice should have it own

1)Project code

2)Project lib

3)Embedded server

4)JVM

5)Operating System(Alpine linux,ubuntu,Debian....)

Example we developed Fare Microservice using spring boot which has its own project code,lib and embedded server so it is partially self contained.

To make it fully self contained we have add JVM and OS to fat jar.

It is possible by creating a container by using Docker.

Docker is a software to create container. Image is created.

A container is a empty box where the buttom layer contain light weight OS, above that JVM, above that Embedded server, above that project libs and finally project code.

code inside the container uses it own OS,JVM,embedded server,project lib and project code but not our OS,JVM,server,etc,....

Example :- if the project in the container contain lambda expression from java 8.

But in my system java 6 is installed then also the code will execute as the project will use the container JVM.

The main advantage to run code inside the container is for seamless or portable execution.

The code inside the container is for seamless or portable execution in different environments like testing,developement,production and cloud environments.

Without containers the code may run in my mechine by not on other machine.because of difference in JVM,difference in server,difference in OS .

Container elemenates this problem .

We have to create separate containers for

1)Fare microservice

2)Booking microservice

3)Search microservice

4)Check-in microservice

And give it to the deployment team or tester team the fat jar. So that we can claim that my microservices are fully self contained.

we never run multiple microservices in a single container.

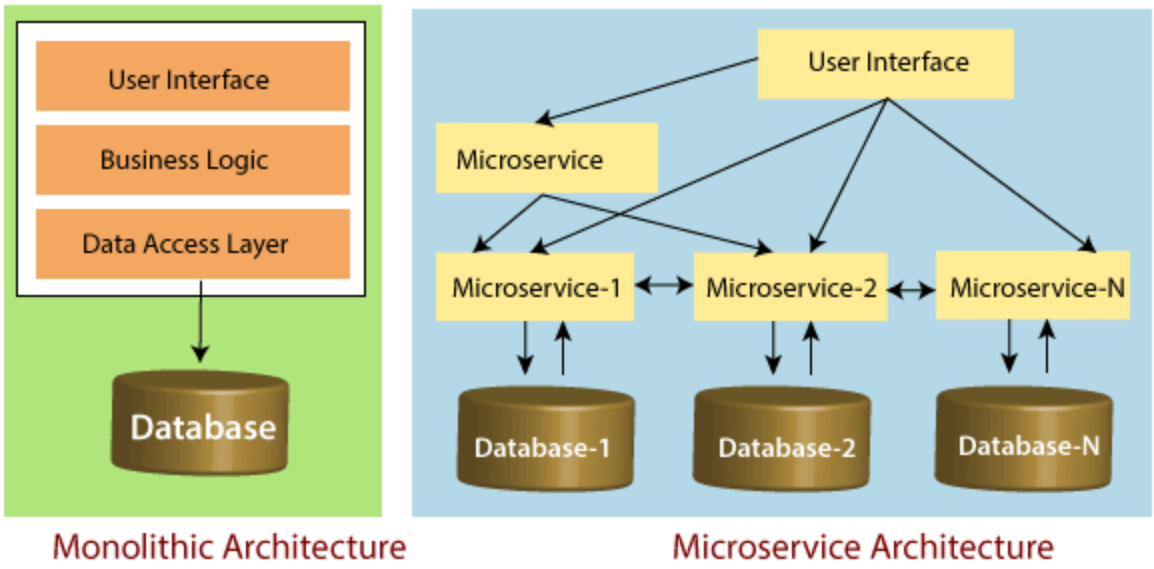
Microservices should independently deployed .it should not dependent on other microservices.

So seperate schema for all the microservices it needed.

Microservices are not invented rather many organizations such as Netflix, Amazon, and eBay successfully used the divide-and-conquer technique to functionally partition their monolithic application into smaller atomic units, each performing a single function.

In below diagram, each layer holds all three business capabilities pertaining to that layer. The presentation layer has web components of all the three modules, the business layer has business components of all the three modules, and the database hosts tables of all the three modules. In most cases, layers are physically spreadable, whereas modules within a layer are hardwired.

Each microservice has its own presentation layer, business layer, and database layer. Microservices are aligned towards business capabilities. By doing so, changes to one microservice don’t impact others.

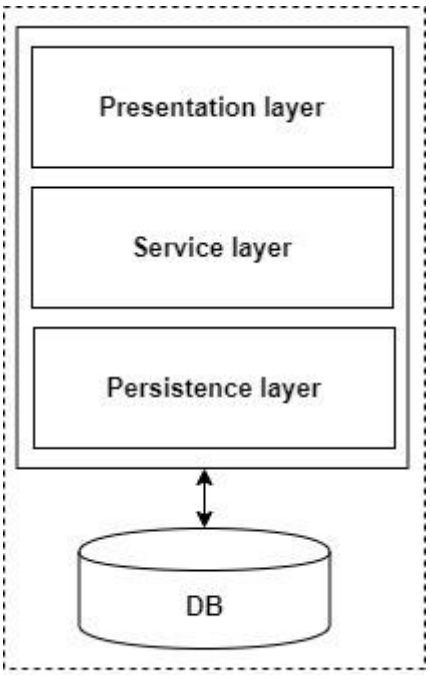


Monolithic vs Microservice Architecture

Benefits of Monolithic Architecture

- Simple to develop.
- Simple to test. For example you can implement end-to-end testing by simply launching the application and testing the UI with Selenium.
- Simple to deploy. You just have to copy the packaged application to a server.

In the early stages of the project it works well and basically most of the big and successful applications which exist today were started as a monolith.



Monolithic architecture

Drawbacks of Monolithic Architecture

- This simple approach has a limitation in size and complexity.
- Application is too large and complex to fully understand and made changes fast and correctly.
- The size of the application can slow down the start-up time.
- You must redeploy the entire application on each update.
- Impact of a change is usually not very well understood which leads to do extensive manual testing.
- Continuous deployment is difficult.
- Monolithic applications can also be difficult to scale when different modules have conflicting resource requirements.
- Another problem with monolithic applications is reliability. Bug in any module (e.g. memory leak) can potentially bring

down the entire process. Moreover, since all instances of the application are identical, that bug will impact the availability of the entire application.

Monolithic applications has a barrier to adopting new technologies. Since changes in frameworks or languages will affect an entire application

In the above figure, each microservice has its own business layer and database. If we change in one microservice, it does not affect the other services. These services communicate with each other by using lightweight protocols such as HTTP or REST or messaging protocols.

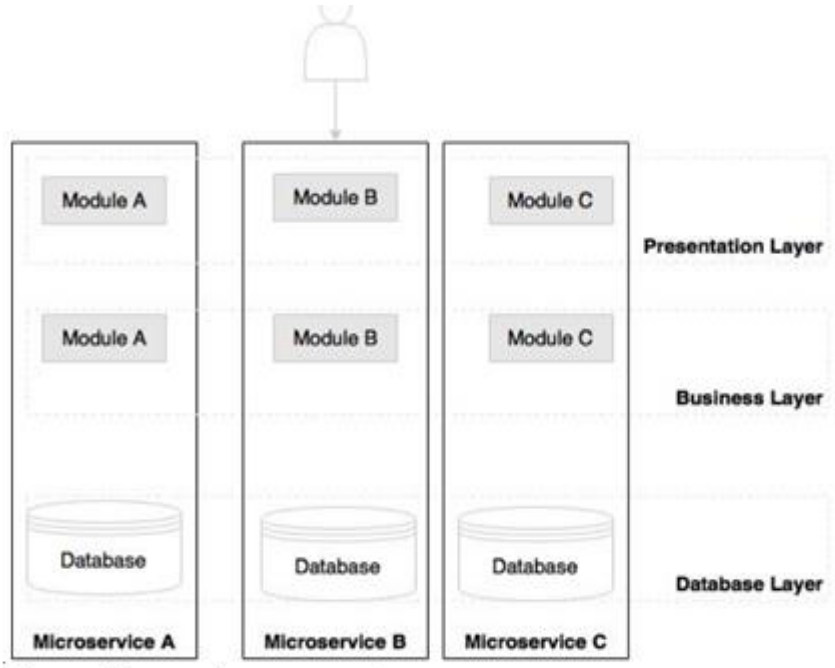


Figure: Microservices approach

Principles of Microservices

- Single Responsibility principle
- Modelled around business domain
- Isolate Failure
- Infrastructure automation
- Deploy independently

Single Responsibility Principle

The single responsibility principle states that a class or a module in a program should have only one responsibility. Any microservice cannot serve more than one responsibility, at a time.

Modeled around business domain

Microservice never restrict itself from accepting appropriate technology stack or database. The stack or database is most suitable for solving the business purpose.

Isolated Failure

The large application can remain mostly unaffected by the failure of a single module. It is possible that a service can fail at any time. So, it is important to detect failure quickly, if possible, automatically restore failure.

Infrastructure Automation

The infrastructure automation is the process of scripting environments. With the help of scripting environment, we can apply the same configuration to a single node or thousands of nodes. It is also known as configuration management, scripted infrastructures, and system configuration management.

Deploy independently

Microservices are platform independent. It means we can design and deploy them independently without affecting the other services.

Advantages of Microservices

- Microservices are self-contained, independent deployment module.
- The cost of scaling is comparatively less than the monolithic architecture.
- Microservices are independently manageable services. It can enable more and more services as the need arises. It minimizes the impact on existing service.
- It is possible to change or upgrade each service individually rather than upgrading in the entire application.
- Microservices allows us to develop an application which is organic (an application which latterly upgrades by adding more functions or modules) in nature.

- It enables event streaming technology to enable easy integration in comparison to heavyweight application communication.

- Microservices follows the single responsibility principle.
- The demanding service can be deployed on multiple servers to enhance performance.
- Less dependency and easy to test.
- Dynamic scaling.
- Faster release cycle.

Disadvantages of Microservices

- Microservices has all the associated complexities of the distributed system.
- There is a higher chance of failure during communication between different services.
- Difficult to manage a large number of services.
- The developer needs to solve the problem, such as network latency and load balancing.
- Complex testing over a distributed environment.

Challenges of Microservices Architecture

Microservice architecture is more complex than the legacy system. The microservice environment becomes more complicated because the team has to manage and support many moving parts. Here are some of the top challenges that an organization face in their microservices journey:

DevOps Culture: Microservices fits perfectly into the DevOps. It provides faster delivery service, visibility across data, and cost-effective data. It can extend their use of containerization switch

from Service-Oriented-Architecture (SOA) to Microservice Architecture (MSA).

Relationship with SOA

SOA and Microservices follow similar concepts i.e., many service characteristics are common in both approaches.

Service-Oriented Architecture (SOA) is an architectural style that supports service orientation. Service orientation is a way of thinking in terms of services and service-based development and the outcomes of services.

A service: Is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit, provide weather data, and consolidate drilling reports)

1) It is self-contained.
2) It may be composed of other services.
3) It is a "black box" to consumers of the service."

We observed similar aspects in microservices as well. Then how microservices differ from SOA?

One of the major differences between Microservices and SOA is in their level of abstraction. While most SOA implementations provide service-level abstraction,

Microservices go further and abstract the realization and execution environment i.e., in SOA development, we may deploy multiple services into the same JEE container. In the microservices approach, each microservice will be built as a fat Jar, embedding all dependencies including web containers and run as a standalone Java process.

Difference between Microservices Architecture (MSA) and Services-Oriented Architecture (SOA)

Microservice Based Architecture (MSA)	Service-Oriented Architecture (SOA)
Microservices uses **lightweight protocols** such as **REST**, and **HTTP**, etc.	SOA supports **multi-message protocols**.
It focuses on **decoupling**.	It focuses on application service **reusability**.
It uses a **simple messaging system** for communication.	It uses **Enterprise Service Bus** (ESB) for communication.
Microservices follows "**share as little as possible**" architecture approach.	SOA follows "**share as much as possible architecture**" approach.
Microservices are much better in **fault tolerance** in comparison to SOA.	SOA is not better in fault tolerance in comparison to MSA.
Each microservice have an **independent** database.	SOA services share the **whole** data storage.
MSA used **modern** relational databases.	SOA used **traditional** relational databases.

SANDIP MOHAPATRA
7760213015