Microservice Architecture is a Service Oriented Architecture. In the microservice architecture, there are a large number of **microservices**. By combining all the microservices, it constructs a big service. In the microservice architecture, all the services communicate with each other.

The Concept of Going Micro

In a service-oriented architecture, entire software packages will be sub-divided into small, interconnected business units. Each of these small business units will communicate to each other using different protocols to deliver successful business to the client. Now the question is, how Microservice Architecture (MSA) differs from SOA? In one word, SOA is a designing pattern and Microservice is an implementation methodology to implement SOA or we can say Microservice is a type of SOA.

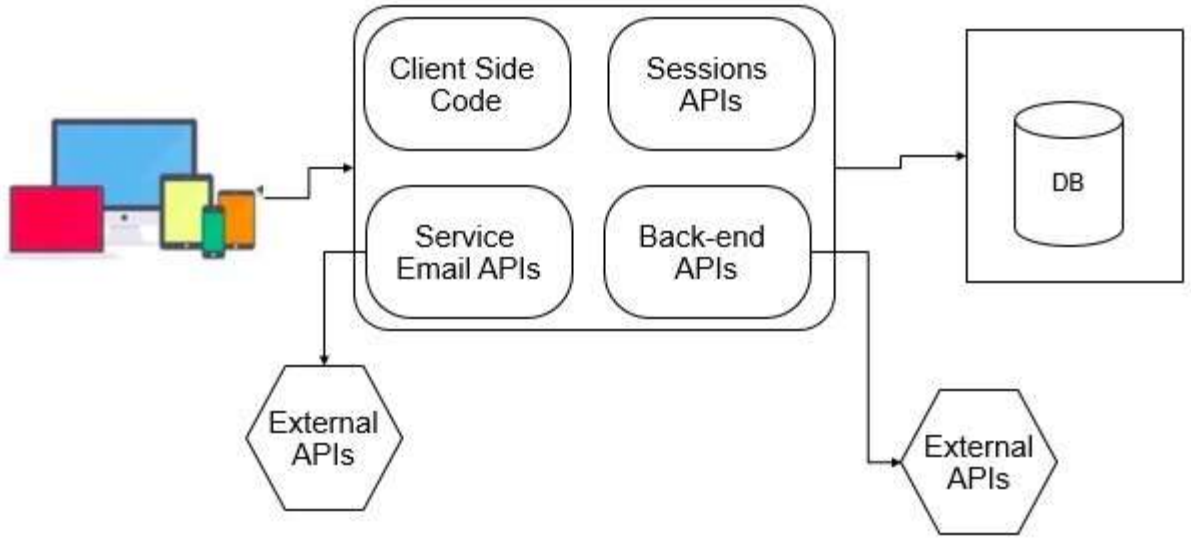
Following are some rules that we need to keep in mind while developing a Microservice-oriented application.

- **Independent** − Each microservice should be independently deployable.
- **Coupling** − All microservices should be loosely coupled with one another such that changes in one will not affect the other.
- **Business Goal** − Each service unit of the entire application should be the smallest and capable of delivering one specific business goal.

Let us consider an example of online shopping portal to understand microservice in depth. Now, let us break this entire E-commerce portal into small business units such as user management, order management, check-in, payment management, delivery management, etc. One successful order needs to proceed through all of these modules within a specific time frame. Following is the consolidated image of different business units associated with one electronic commerce system.

Each of these business modules should have its own business logic and stakeholders. They communicate with other third party vendor softwares for some specific needs, and also with each other. For example, order management may communicate with user management to get user information.

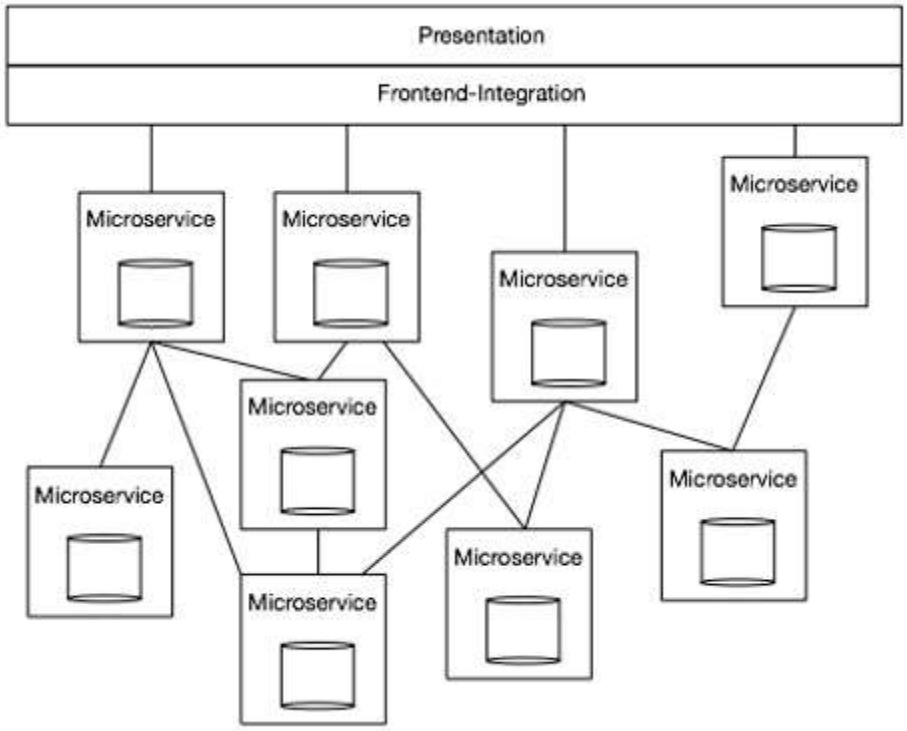
Now, considering you are running an online shopping portal with all of these business units mentioned earlier, you do need some enterprise level application consisting of different layers such as front-end, back-end, database, etc. If your application is not scaled and completely developed in one single war file, then it will be called as a typical monolithic application. According to IBM, a typical monolithic application should possess the following module structure internally where only one endpoint or application will be responsible to handle all user requests.



In the above image, you can see different modules such as Database for storing different users and business data. At the front-end, we have different device where we usually render user or business data to use. In the middle, we have one package that can be a deployable EAR or WAR file that accepts request form the users end, processes it with the help of the resources, and renders it back to the users. Everything will be fine until business wants any changes in the above example.

Consider the following scenarios where you have to change your application according to the business needs.

Business unit needs some changes in the “Search” module. Then, you need to change the entire search process and redeploy your application. In that case, you are redeploying your other units without any changes at all.



Now again your business unit needs some changes in “Check out” module to include “wallet” option. You now have to change your “Check out” module and redeploy the same into the server. Note, you are redeploying the different modules of your software packages, whereas we have not made any changes to it. Here comes the concept of service-oriented architecture more specific to Microservice architecture. We can develop our monolithic application in such a manner that each and every module of the software will behave as an independent unit, capable of handling a single business task independently.

Consider the following example.

In the above architecture, we are not creating any ear file with compact end-to-end service. Instead, we are dividing different parts of the software by exposing them as a service. Any part of the software can easily communicate with each other by consuming respective services. That's how microservice plays a great role in modern web application.

Let us compare our shopping cart example in the line of microservice. We can break down our shopping cart in the different modules such as “Search”, ”Filter”, “Checkout”, “Cart”, “Recommendation”, etc. If we want to build a shopping cart portal then we have

to build the above-mentioned modules in such a manner that they can connect to each other to give you a 24x7 good shopping experience.

What are Microservices

"Microservices are the small services that work together."

"The microservice architectural style is an approach to develop a single application as a suite of small services. Each microservice runs its process and communicates with lightweight mechanisms. These services are built around business capabilities and independently developed by fully automated deployment machinery."

There is a bare minimum of centralized management of these services, which may be written in different programming language and use different data storage technologies. programming language and use different data storage technologies.

Points to remember

- These are the services which are exposed by REST.
- These are small well-chosen deployable units.
- The services must be cloud-enabled.

The microservice defines an approach to the architecture that divides an application into a pool of loosely coupled services that implements business requirements. It is next to **Service-Oriented Architecture (SOA)**. The most important feature of the microservice-based architecture is that it can perform **continuous delivery** of a large and complex application.

Microservice helps in breaking the application and build a logically independent smaller applications. For example, we can build a cloud application with the help of Amazon AWS with minimum efforts.

Principles of Microservices

There are the following 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 agnostic. 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 interposes 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:

- Bounded Context
- Dynamic Scale up and Scale Down
- Monitoring
- Fault Tolerance
- Cyclic dependencies
- DevOps Culture

Bounded context: The bounded context concept originated in Domain-Driven Design (DDD) circles. It promotes the Object model first approach to service, defining a data model that service is responsible for and is bound to. A bounded context clarifies, encapsulates, and defines the specific responsibility to the model. It ensures that the domain will not be distracted from the outside. Each model must have a context implicitly defined within a sub-domain, and every context defines boundaries.

In other words, the service owns its data and is responsible for its integrity and mutability. It supports the most important feature of microservices, which is independence and decoupling.

Monolithic architecture is built as one large system and is usually one code-base. Monolithic application is tightly coupled and entangled as the application evolves, making it difficult to isolate services for purposes such as independent scaling or code maintainability.

It extremely difficult to change technology or language or framework because everything is tightly coupled and depend on each other.

Microservices architecture is built as small independent module based on business functionality. In microservices application, each project and services are independent from each other at the code level. Therefore it is easy to configure and deploy completely and also easy to scale based on demand.

Sr. No.	Key	Monolithic architecture	Microservices architecture
1	Basic	Monolithic architecture is built as one large system and is usually one code-base	Microservices architecture is built as small independent module based on business functionality
2	Scale	It is not easy to scale based on demand	It is easy to scale based on demand.
3	Database	It has shared database	Each project and module has their own database
4	Deployment	Large code base makes IDE slow and build time gets increase.	Each project is independent and small in size. So overall build and development time gets decrease.
5	Tightly Coupled	It extremely difficult to change technology or	Easy to change technology or framework

Sr. No.	Key	Monolithic architecture	Microservices architecture
	and Loosely coupled	language or framework because everything is tightly coupled and depend on each other	because every module and project is independent

Eureka naming server

Eureka naming server is a REST-based server that is used in the **AWS Cloud** services for load balancing and failover of middle-tier services.