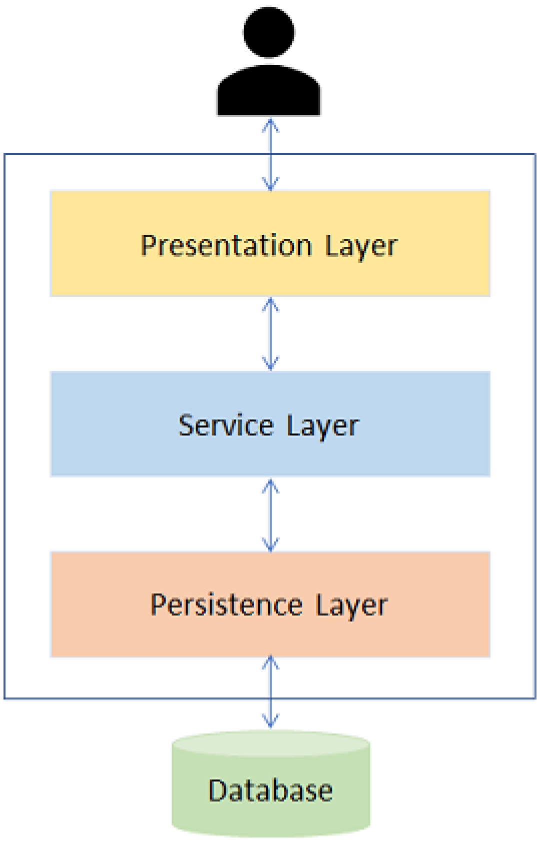
Consider an online shopping application with the following functionalities:

1. Search products
2. Place order
3. View all orders

The most common approach to develop this application is to implement all these functionalities in a single application. This application is then deployed as one application for all different types of users. This type of architecture where all the functionalities are implemented in one single application is called as **Monolithic Architecture**. In this architecture the application is divided into different layers. Some common layers in this architecture are as follows:

- **Presentation Layer** — It responsible for handling HTTP requests and sending respose.
- **Service Layer** — In this layer business logic of the application is implemented.
- **Persistence layer** — In this layer logic for accessing the database is implemented.



The applications developed using this architecture are easy to develop, test and deploy. However, there are certain issues that tag along with monolithic applications. As the application becomes larger and complex, we will face difficulties in the following aspects:

1. Deployment will take a long time.

2. Scalability is an issue as it is not possible to scale any single functionality alone. For example, the 'Search' functionality maybe used more often than the 'Book' functionality. Hence, we would like to scale the 'Search' functionality and not the 'Book' functionality. This is not possible when we go with the monolithic approach.
3. Failure of a single functionality will lead to failure of entire application in monolithic architecture.
4. New technologies or frameworks cannot be used in the existing application. If new technology is needed, then complete re-write must be done.
5. It is not very reliable as a single bug in any module can bring down the entire application.

How do we resolve these issues?

The answer is **Microservices**. It is another architecture of application development. Now let us explore it in detail.

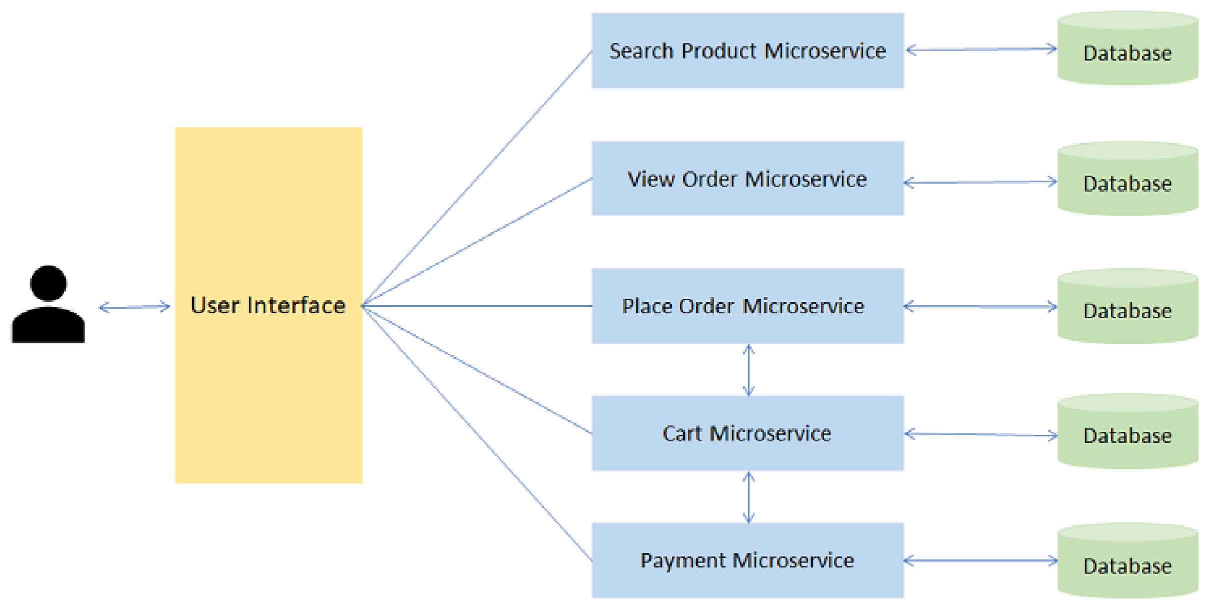
What are Microservices?

We have learned that in monolithic architecure all the fuctionalities are implemented in one single application and then this application is deployed. But it has some drawbacks which can be addressed if we develop the application using another architecure called as **microservices**. In this architecture we split the application into a set of smaller services instead of building a single monolithic application. Each of these services implements specific business requirement and can communicate with each other through well defined interfaces. These services have their own database and can be deployed independently.

Consider the online shopping application which we have developed using monolithic architecure. Now if you have do develop same application using microservices, then you need to split the business requirements or functonalities into several services communicating with each other and having its own database. You can have following services for this application:

- Search Product Microservice - Responsible for seaching a product.
- View Order Microservice - View orders placed by customer.
- Place Order Microservice - Takes an order and process it.
- Cart Microservice - Manage user cart, this service can utilize Catalog service as a data source.
- Payment Microservice - Manage payments.

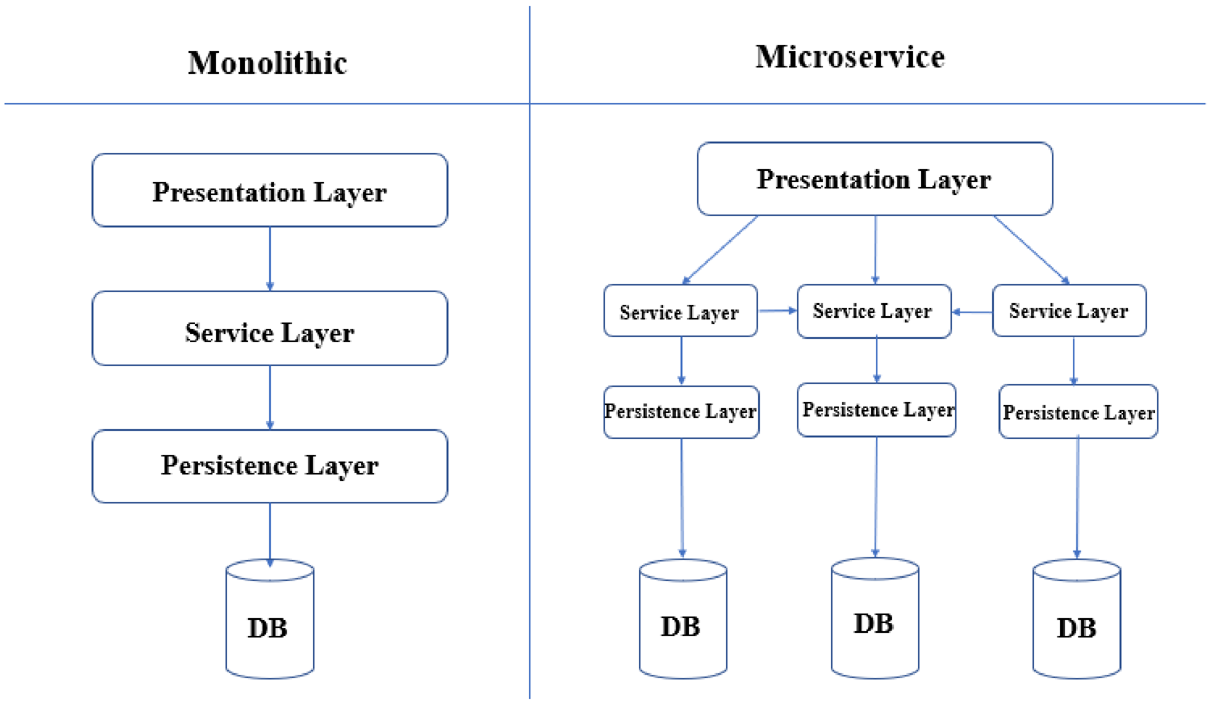
Each of these microservices also interact with each other to transfer data which is required by another microservice. For example, Place Order microservice will interact with Card and Order Microservice.



Now that we have seen a real-life example of Microservice, let us understand why we should choose Microservices for developing enterprise applications.

Monolithic v/s Microservices

You have learnt the architectural differences between Monolithic Applications and Microservices. Now let us see how functionally wise they are different.

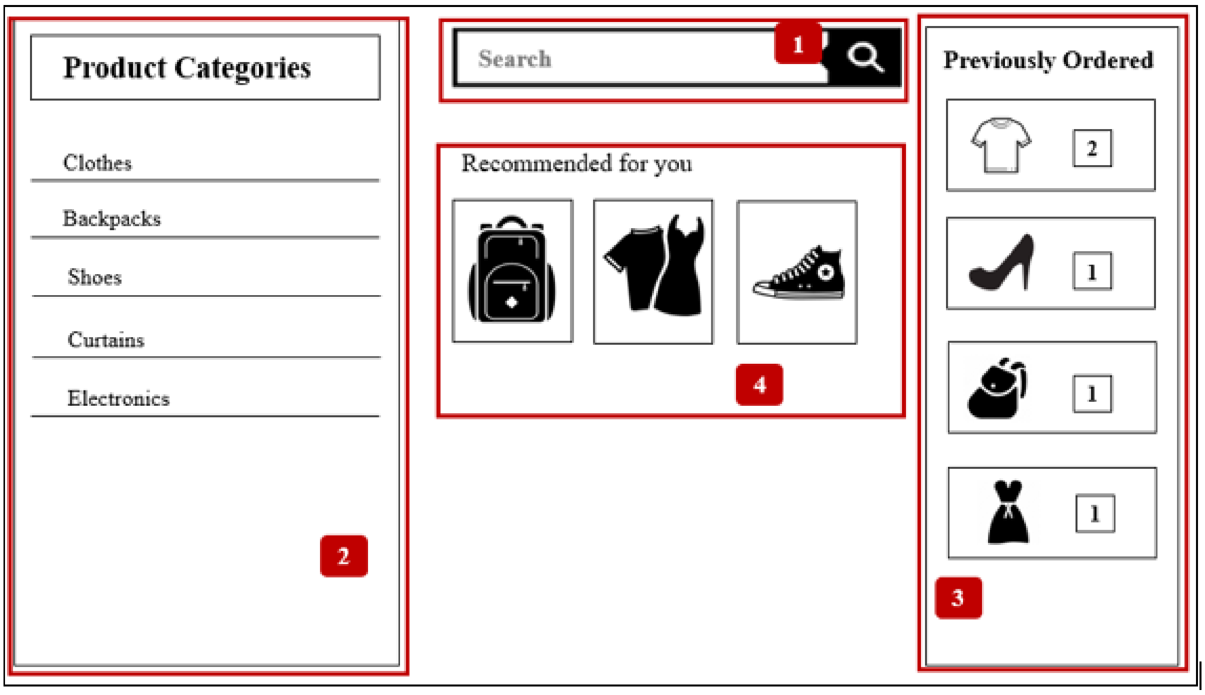


Parameter	Monolithic	Microservice
Development	Needs to be developed in a single language	Each service can be written in different languages
Testing	For any change, the entire application needs to be tested again	Only the modified service needs to be tested again
Runtime	Monolithic application runs as a single process	Each service runs its own process
Packaging	Packaged as a single JAR/WAR file	Each service is packaged as single JAR/WAR file
Scalability	Entire application needs to be scaled or replicated on multiple servers	Only the service that requires scaling, can be scaled
Minor Changes	For any modification. entire application needs to be re-built and re-deployed	Only the modified service needs to be re-built and re-deployed

Now we will understand more about Microservices with an example.

Example of Microservice architecture

An online shopping application can be divided into a number of microservices based on the functionality and business requirements.



This application can be divided into the following microservices:

1. **Product Microservice**: This microservice can be used to help the User search for any specific product that they are looking for.

2. **Product Categories Microservice**: This microservice shows all product categories available to the User.
3. **Orders Microservice**: This Microservice is responsible for placing orders and the User can view the order history as well.
4. **Recommended Products Microservice**: This microservice shows a list of recommended products to the User based on purchase history.

Now that we have seen a real-life example of Microservice, let us understand why we should choose Microservices for developing enterprise applications.

Top Reasons for choosing Microservice

The following are the top reasons for choosing Microservice architecture:

- **Easy deployment**: A large code base will make the IDE slow and build time increases. In Microservice architecture, each project is independent and small in size. So overall build and development time gets reduced.
- **Small teams**: Microservices have single-purpose design, which means they can be built and maintained by smaller teams. Moreover, each team can be cross-functional i.e each team can have their own developers, testers and operations team and can be responsible for a single microservice.
- **Loose coupling**: It extremely difficult to change technology or language or framework when everything is tightly coupled and dependent on each other. In microservices, it is easy to change technology or framework because every module and project is independent and loosely coupled.
- **Domain-driven design**: Although microservices are supposed to be small in size, it is not the only criteria. The services should have well-defined boundaries centered around business requirements. Domain-driven design, helps design software systems based on the underlying model of the business domain, which is the requirement of microservices.

Microservice Usage Scenarios

As we have seen, there are multiple reasons for choosing Microservice architecture. The following are the scenarios in which microservices can be used.

1. Migrating a monolithic application due to improvements required in scalability, manageability and agility.
2. Rewriting a heavily used legacy application. (A legacy application is a software program that is outdated or obsolete.)
3. Highly agile applications due to domain-driven design of microservices.
4. Applications that demand speed of delivery as microservices are easy to build and maintain.
5. New product development where a new product is conceived and brought to market.

Does this mean we should use microservice architecture always?

The answer is No.

If we look at the evolution of software, we started with unstructured code. Then we moved on to packages for codes, which slowly moved on to creating libraries for reuse. Then we had multiple applications and multiple libraries which had services added to them.

So, when should we choose Microservices and when should we choose Monolithic applications

We should choose Monolithic when:

1. We have little knowledge of the market
2. The application we are developing is small

We should choose Microservices when:

1. We have more knowledge of the market
2. The application being developed is a large enterprise application and needs to be highly scalable.

Advantages of Microservices

Advantages and Disadvantages of Microservices

There are some advantages of using microservices.

1. **Scalability**: It is easier to scale only the required service based on demand.
2. **Fault Isolation**: Failure in one service does not affect the working of other services. So the entire application does not fail due to failure of single functionality.
3. **Speed of deployment**: As services are small, it can be built and deployed faster.
4. **Freedom of technology**: Each service can be written in different technology, using different frameworks based on requirement.
5. **Autonomous teams**: Microservices grant the developers more independence to work autonomously and make technical decisions quickly in smaller groups.

Although microservices have advantages, there are a few disadvantages as well. The following are the disadvantages of using microservices.

1. **Performance**: As we have a more distributed and complex application, performance could be affected.
2. **Maintenance**: An application can have hundreds of microservices and maintaining all of them can be a challenge. This can be overcome by de-centralized management.

3. **Infrastructure**: Initially, microservices require a huge infrastructure setup.
4. **Cost**: Due to huge infrastructure setup huge cost may also be incurred. Also, services need to communicate with each other. This can increase network latency and processing costs.