Add instructor notes here.

Objective

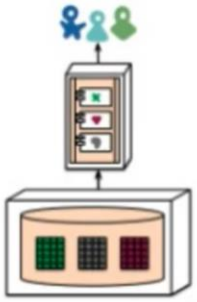
- Monolithic Architecture
- SOA
- MicroServices Architecture

Monolithic Architecture

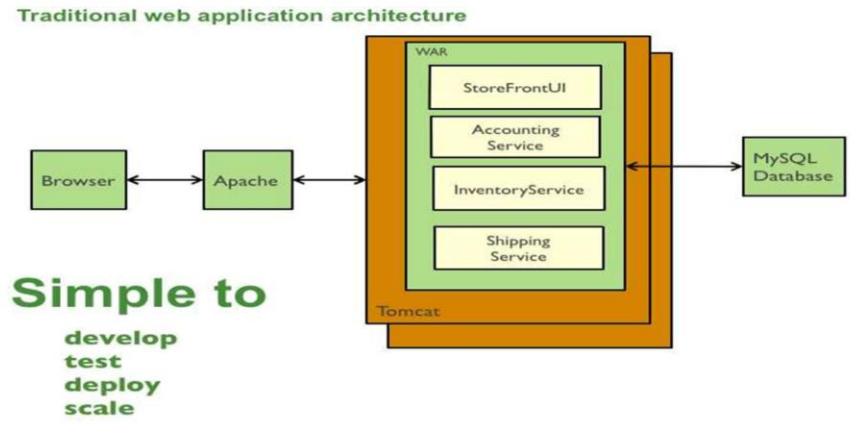
- In software engineering, a monolithic application describes a single-tiered software application in which the user interface and data access code are combined into a single program from a single platform.
- A monolithic application is self-contained, and independent from other computing applications. The design philosophy is that the application is responsible not just for a particular task, but can perform every step needed to complete a particular function.
- Today, some personal finance applications are monolithic in the sense that they help the user carry out a complete task, end to end, and are private data silos rather than parts of a larger system of applications that work together. Some word processors are monolithic applications. These applications are sometimes associated with mainframe computers.
- In software engineering, a monolithic application describes a software application which is designed without modularity. Modularity is desirable, in general, as it supports reuse of parts of the application logic and also facilitates maintenance by allowing repair or replacement of parts of the application without requiring wholesale replacement.

Monolithic Architecture

- Monolithic architecture is very similar to joint families. All components are present together in one box and all data stored in one database.



Monolithic Architecture



Forces
There is a team of developers working on the application
New team members must quickly become productive
The application must be easy to understand and modify
You want to practice continuous deployment of the application
You must run multiple instances of the application on multiple machines in order to satisfy scalability and availability requirements
You want to take advantage of emerging technologies (frameworks, programming languages, etc)
Solution
Build an application with a monolithic architecture. For example:
a single Java WAR file.
a single directory hierarchy of Rails or NodeJS code
Example
Let's imagine that you are building an e-commerce application that takes orders from customers, verifies inventory and available credit, and ships them. The application consists of several components including the StoreFrontUI, which implements the user interface, along with some backend services for checking credit, maintaining inventory and shipping orders.
The application is deployed as a single monolithic application. For example, a Java web application consists of a single WAR file that runs on a web container such as Tomcat. A Rails application consists of a single directory hierarchy deployed using either, for example, Phusion Passenger on Apache/Nginx or JRuby on Tomcat. You can run multiple instances of the application behind a load balancer in order to scale and

improve availability.

Service-oriented architecture(SOA)

- Service-oriented architecture (SOA) is a style of software design where services are provided to the other components by application components, through a communication protocol over a network. An SOA service is a discrete unit of functionality that can be accessed remotely and acted upon and updated independently, such as retrieving a credit card statement online.
- SOA is also intended to be independent of vendors, products and technologies

A service has four properties according to one of many definitions of SOA:

- It logically represents a business activity with a specified outcome.
- It is self-contained.
- It is a black box for its consumers, meaning the consumer does not have to be aware of the service's inner workings.
- It may consist of other underlying services

Service-oriented architecture principles are made up of nine main elements:
1. Standardized Service Contract where services are defined making it easier for client applications to understand the purpose of the service.
2. Loose Coupling is a way to interconnecting components within the system or network so that the components can depend on one another to the least extent acceptable. When a service functionality or setting changes there is no downtime or breakage of the application running.
3. Service Abstraction hides the logic behind what the application is doing. It only relays to the client application what it is doing, not how it executes the action.
4. Service Reusability divides the services with the intent of reusing as much as possible to avoid spending resources on building the same code and configurations.
5. Service Autonomy ensures the logic of a task or a request is completed within the code.
6. Service Statelessness whereby services do not withhold information from one state to another in the client application.

Service-oriented architecture(SOA)

- Different services can be used in conjunction to provide the functionality of a large software application.
- A principle SOA shares with modular programming.
- Service-oriented architecture integrates distributed, separately maintained and deployed software components. It is enabled by technologies and standards that facilitate components' communication and cooperation over a network, especially over an IP network.
- SOA is related to the idea of an application programming interface (API), an interface or communication protocol between different parts of a computer program intended to simplify the implementation and maintenance of software. An API can be thought of as the service, and the SOA the architecture that allows the service to operate.

7. Service Discoverability allows services to be discovered via a service registry.
8. Service Composability breaks down larger problems into smaller elements, segmenting the service into modules, making it more manageable.
9. Service Interoperability governs the use of standards (e.g. XML) to ensure larger usability and compatibility.

How Does Service-Oriented Architecture Work?

A service-oriented architecture (SOA) works as an components provider of application services to other components over a network. Service-oriented architecture makes it easier for software components to work with each other over multiple networks.

Service-oriented architecture is implemented with web services (based on WSDL and SOAP), to be more accessible over standard internet protocols that are on independent platforms and programming languages.

Service-oriented architecture has 3 major objectives all of which focus on parts of the application cycle:

Service-oriented architecture(SOA)

Service-oriented architecture (SOA) is a software architecture style that supports and distributes application components that incorporates discovery, data mapping, security and more. Service oriented architecture has two main functions:

1) Create a architectural model that defines goals of applications and methods that will help achieve those goals.

2) Define implementations specifications linked through WSDL (Web Services Description Language) and SOAP (Simple Object Access Protocol) specifications.

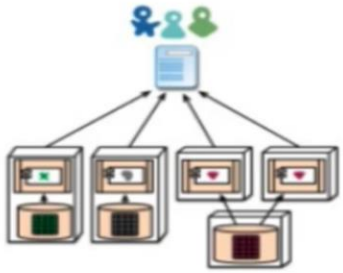
1) Structure process and software components as services – making it easier for software developers to create applications in a consistent way.
2) Provide a way to publish available services (functionality and input/output requirements) – allowing developers to easily incorporate them into applications.
3) Control the usage of these services for security purposes – mainly around the components within the architecture, and securing the connections between those components.

Microservices

- Microservices have emerged from concepts like domain-driven design, Continuous delivery, On-demand virtualization, Infrastructure automation and Small autonomous teams.

"Microservices are small, autonomous services that work together."

- Embracing fine-grained, microservice architectures, they can deliver software faster and embrace newer & heterogeneous technologies.
- Microservices make organization agile and allowing to respond faster to the inevitable changes that are constant.



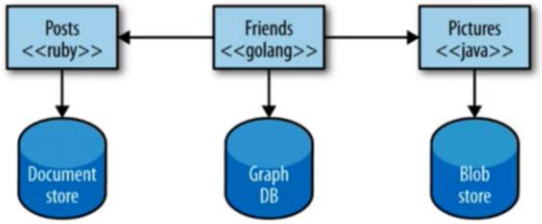
Microservices

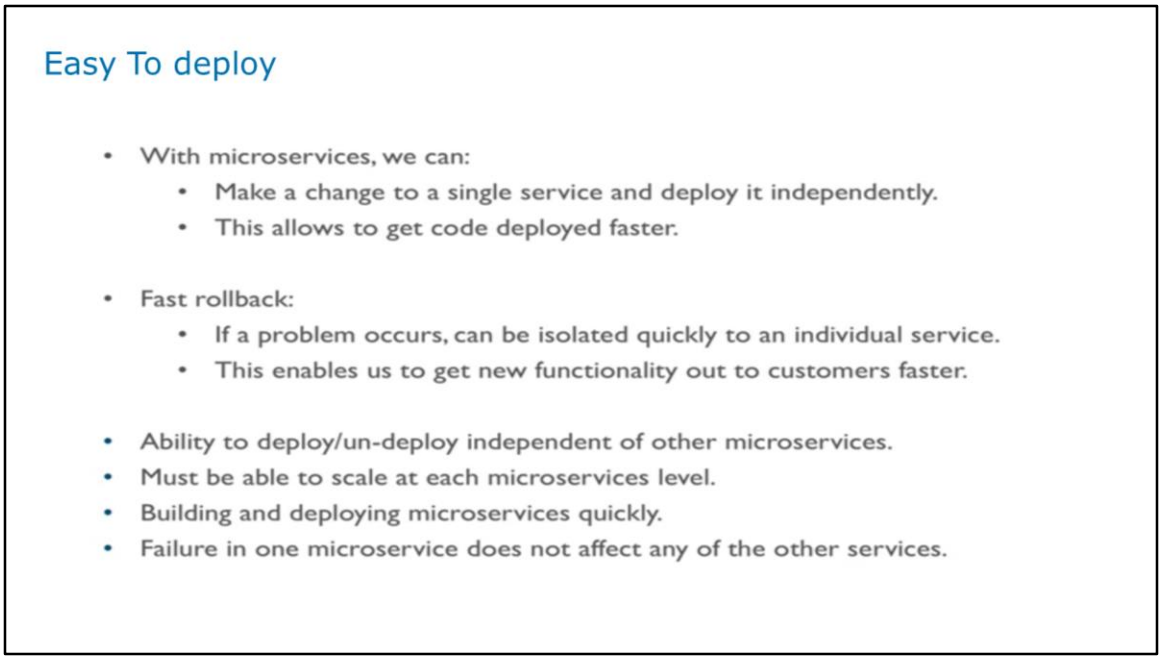
- Microservices are driven from business domains and needs. It breaks the domain functions in smaller independent components. Technology helps in achieving these components to be developed , deployed and function independently.
- Since they are small and independent this makes them easy to manage and changes can be done quickly.
- The Learning curve for such smaller components is very short, new developers can be on boarded and start to contribute.

When to use the microservice architecture?
One challenge with using this approach is deciding when it makes sense to use it. When developing the first version of an application, you often do not have the problems that this approach solves. Moreover, using an elaborate, distributed architecture will slow down development. This can be a major problem for startups whose biggest challenge is often how to rapidly evolve the business model and accompanying application. Using Y-axis splits might make it much more difficult to iterate rapidly. Later on, however, when the challenge is how to scale and you need to use functional decomposition, the tangled dependencies might make it difficult to decompose your monolithic application into a set of services.

Heterogeneity of Technology

- With a system composed of multiple, collaborating services, different technologies can be chosen inside each one depending on the best fitment.
- This enables organizations to pick the right tool for each job instead of having to select a more standardized, one-size-fits-all approach.





How to decompose the application into services?
Another challenge is deciding how to partition the system into microservices. This is very much an art, but there are a number of strategies that can help:
Decompose by business capability and define services corresponding to business capabilities.
Decompose by domain-driven design subdomain.
Decompose by verb or use case and define services that are responsible for particular actions. e.g. a Shipping Service that's responsible for shipping complete orders.
Decompose by by nouns or resources by defining a service that is responsible for all operations on entities/resources of a given type. e.g. an Account Service that is responsible for managing user accounts.
Ideally, each service should have only a small set of responsibilities. (Uncle) Bob Martin talks about designing classes using the Single Responsibility Principle (SRP). The SRP defines a responsibility of a class as a reason to change, and states that a class should only have one reason to change. It make sense to apply the SRP to service design as well.
Another analogy that helps with service design is the design of Unix utilities. Unix provides a large number of utilities such as grep, cat and find. Each utility does exactly one thing, often exceptionally well, and can be combined with other utilities

using a shell script to perform complex tasks.

Principle of MicroService

- Culture of Automation
- Hide Implementation Detail
- Decentralized all things
- Deploy Independently
- Customer First
- Isolate Failure
- Highly Observable

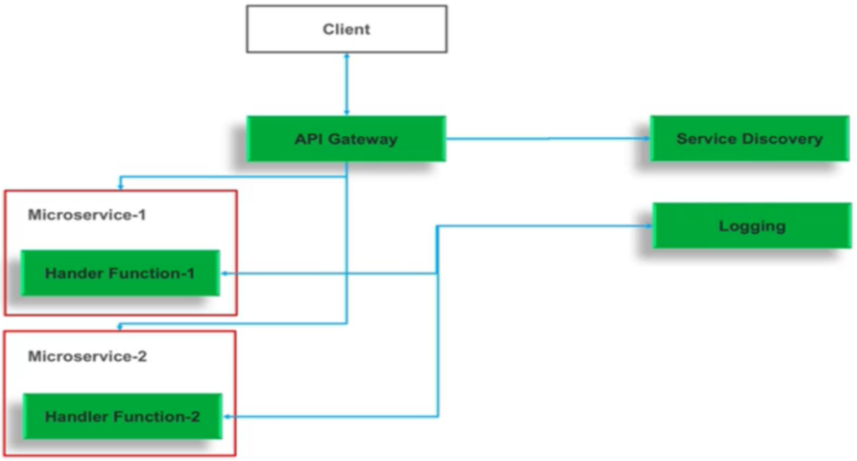
Developer Independence
With small teams working on a microservice, it contributes to much more developer freedom and independence. Small teams that are working in parallel can iterate faster than large teams. They can also scale the services on their own without having to wait for a larger and more complex team.
Isolation and Resilience
Another big advantage of microservices is their qualities of isolation and resilience. If one of the components should fail, due to issues including the technology becomes outdated or the code in the service cannot be developed any further, developers are able to spin up another while the rest of the application continues to function independently. This capability gives developers the freedom to develop and deploy services as needed without having to wait on decisions concerning the entire application.
Scalability
Due to the fact that microservices are made of much smaller components, they are able to take up fewer resources and therefore more easily scale to meet increasing demand of that specific component only. As a result of their isolation, microservices can properly function even during large changes in size and volume, making it an ideal method for enterprises dealing with a wide range of platforms and devices.
Autonomously Developed
As opposed to monoliths, individual components are much easier to fit into continuous delivery pipelines and complex deployment scenarios. Only the pinpointed service needs to be modified and redeployed when a change is needed, and if a service should fail, the others will continue to function independently. Outside of the obvious benefits this offers the system, this autonomous nature is also beneficial to the team as it enables scaling and development without requiring much coordination between other teams, particularly an advantage as companies become more distributed and workers more remote.
Relationship to the Business
Microservice architectures are split along business domain boundaries, organized around capabilities such as logistics, billing, etc. This increases independence and understanding across the organization as different teams

are able to utilize a specific product and then own and maintain it for its lifetime.

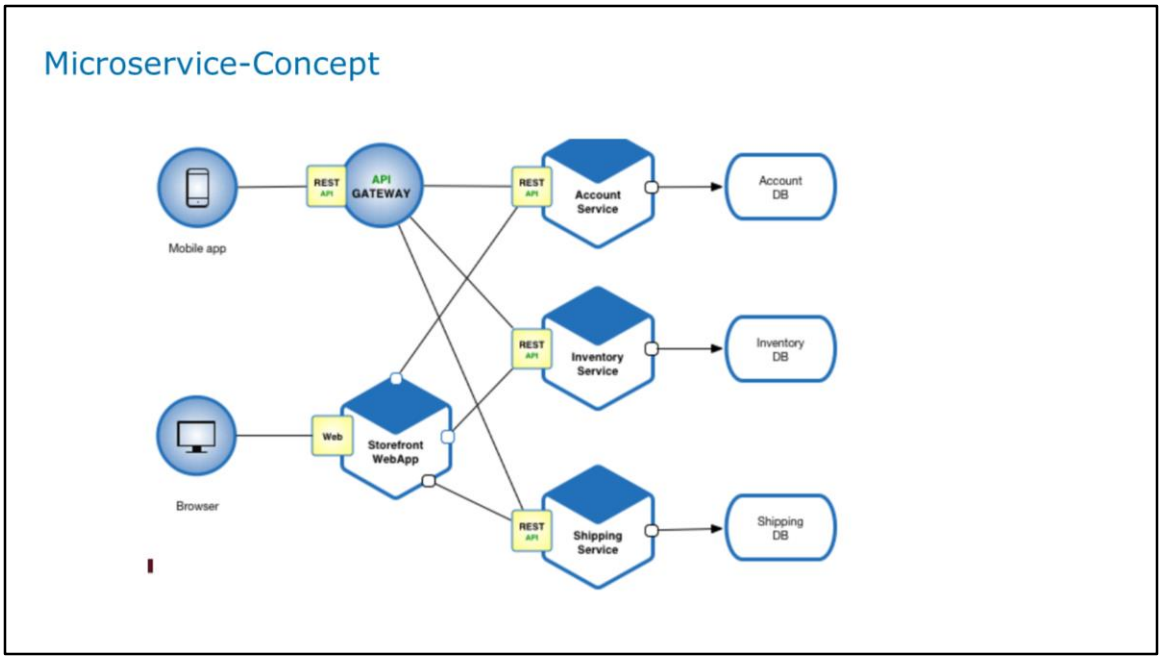
Evolutionary

A final benefit of microservice architecture is the fact that it is highly evolutionary. Microservices are an excellent option for situations where developers can't fully predict what devices will be accessed by the application in the future, and they allow quick and controlled changes to the software without slowing the application as a whole.

Microservice Ecosystem



Popular Microservices Adopter



Fictitious e-commerce application

Let's imagine that you are building an e-commerce application that takes orders from customers, verifies inventory and available credit, and ships them. The application consists of several components including the StoreFrontUI, which implements the user interface, along with some backend services for checking credit, maintaining inventory and shipping orders. The application consists of a set of services.

Benefit & Issue

Benefits

- Enables the continuous delivery and deployment of large, complex applications.
- Each microservice is relatively small.
- Improved fault isolation.

Issue

- When to use the microservice architecture?
- How to decompose the application into services?
- How to maintain data consistency?
- How to implement queries?

Benefits

This solution has a number of benefits:

Enables the continuous delivery and deployment of large, complex applications.

Improved maintainability - each service is relatively small and so is easier to understand and change

Better testability - services are smaller and faster to test

Better deployability - services can be deployed independently

It enables you to organize the development effort around multiple, autonomous teams. Each (so called two pizza) team owns and is responsible for one or more services. Each team can develop, test, deploy and scale their services independently of all of the other teams.

Each microservice is relatively small:

Easier for a developer to understand

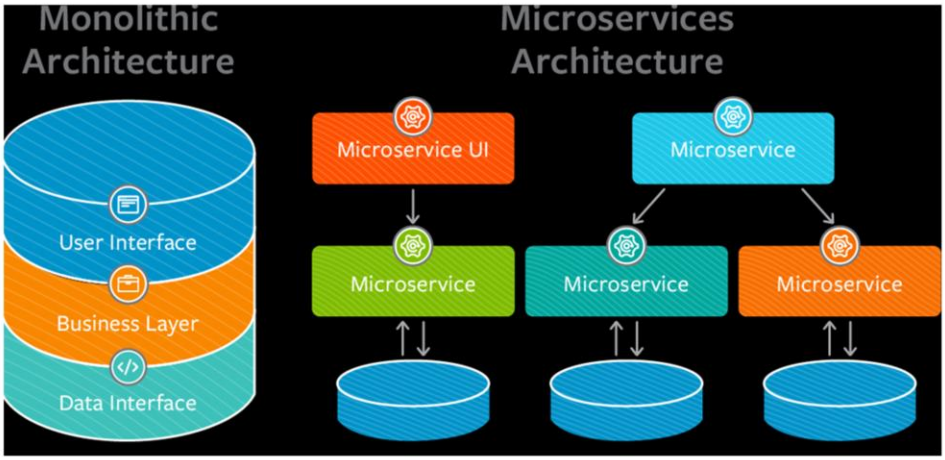
The IDE is faster making developers more productive

The application starts faster, which makes developers more productive, and speeds up deployments

Improved fault isolation. For example, if there is a memory leak in one service then only that service will be affected. The other services will continue to handle requests. In comparison, one misbehaving component of a monolithic architecture can bring

down the entire system.
Eliminates any long-term commitment to a technology stack. When developing a new service you can pick a new technology stack. Similarly, when making major changes to an existing service you can rewrite it using a new technology stack.

Monolithic & Microservices Architecture



Drawbacks

This solution has a number of drawbacks:

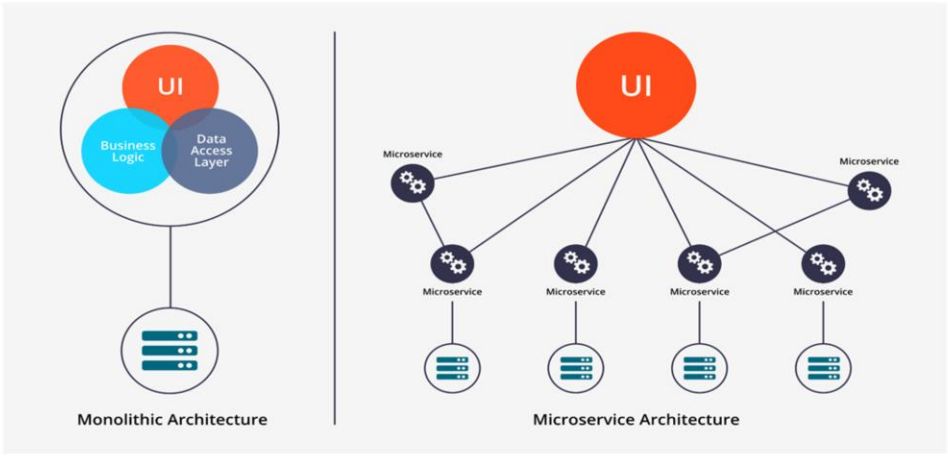
Developers must deal with the additional complexity of creating a distributed system:

- Developers must implement the inter-service communication mechanism and deal with partial failure
- Implementing requests that span multiple services is more difficult
- Testing the interactions between services is more difficult
- Implementing requests that span multiple services requires careful coordination between the teams
- Developer tools/IDEs are oriented on building monolithic applications and don't provide explicit support for developing distributed applications.

Deployment complexity. In production, there is also the operational complexity of deploying and managing a system comprised of many different services.

Increased memory consumption. The microservice architecture replaces N monolithic application instances with NxM services instances. If each service runs in its own JVM (or equivalent), which is usually necessary to isolate the instances, then there is the overhead of M times as many JVM runtimes. Moreover, if each service runs on its own VM (e.g. EC2 instance), as is the case at Netflix, the overhead is even higher.

Monolithic & Microservices Architecture



The monolithic architecture pattern is the traditional architectural style that many systems utilize, with the monolith application built as a single, autonomous unit. While this style has been an integral part of many businesses, its numerous limitations and issues are motivating more and more to make the switch to microservices.

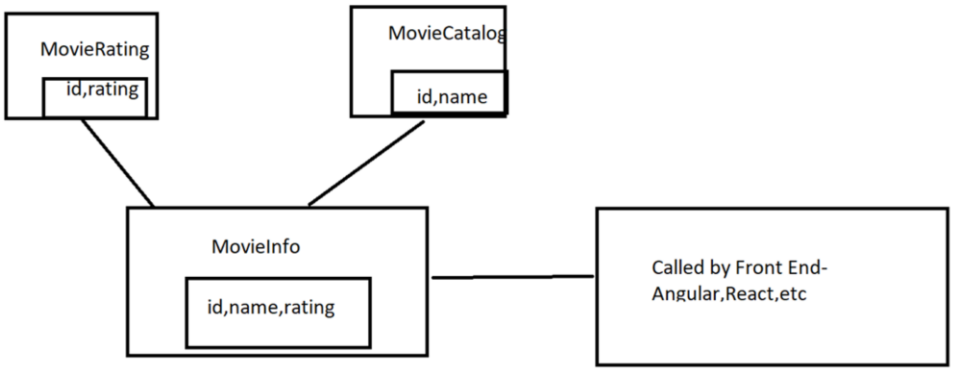
Monolithic structures make any changes to the application extremely slow as it often affects the entire system. It can require a completely rebuilt and deployed version of software whenever a modification is made to a small section of code. If developers wish to scale certain functions of an application, they must scale the entire application, further complicating changes and updates. Microservices help to solve these challenges and more.

SOA & Microservices

SOA is like an orchestra where each artist is performing with his/her instrument while the music director guides them all.

With Microservices each dancer is independent and know what they need to do. If they miss some steps they know how to get back on the sequence.

Microservice -Design



Demo

MovieCatalogService
MovieCatalogService
MovieRating

Which is the following Order following concepts are introduced ?

a. MicroServices,SOA , Distributed
b. SOA,Distributed, Microservices
c. Distributed,SOA,Microservices