Microservices

484. What is a Microservice?

A Microservice is a small and autonomous piece of code that does one thing very well. It is focused on doing well one specific task in a big system.
It is also an autonomous entity that can be designed, developed and deployed independently.

Generally, it is implemented as a REST service on HTTP protocol, with technology-agnostic APIs.

Ideally, it does not share database with any other service.

485. What are the benefits of Microservices architecture?

Microservices provide many benefits. Some of the key benefits are:

1. Scaling: Since there are multiple Microservices instead of one monolith, it is easier to scale up the service that is being used more. Eg. Let say, you have a Product Lookup service and Product Buy service. The frequency of Product Lookup is much higher than Product Buy service. In this case, you can just scale up the Product Lookup service to run on powerful hardware with multiple servers. Meanwhile, Product Buy service can remain on less powerful hardware.
2. Resilience: In Microservice architecture, if your one service goes down, it may not affect the rest of the system. The other parts can keep functioning, business as usual (BAU). Eg. Let say, you have Product Recommendation service and Product Buy service. If Product Recommendation service goes down, the Product Buy service can still keep running.
3. Technology Mix: With so many changes in technology everyday, you can keep using the latest technology for your new Microservices. You can adopt new technologies with less risk compared to Monolithic architecture. This is one of the best benefits of Microservices architecture.
4. Reuse: Microservices help you in reusing the lessons learnt from one service to another.
5. Easy Deployment: Microservices architecture, if done correctly, helps in making the deployment process smooth. If anything goes wrong, it can be rolled back easily and quickly in Microservices.

486. What is the role of architect in Microservices architecture?

Architects, in Microservices architecture, play the role of Town planners. They decide in broad strokes about the layout of the overall software system.

They help in deciding the zoning of the components. They make sure components are mutually cohesive but not tightly coupled. They need not worry about what is inside each zone.
Since they have to remain up to date with the new developments and problems, they have to code with developers to learn the challenges faced in day-to-day life.
They can make recommendations for certain tools and technologies, but the team developing a micro service is ultimately empowered to create and design the service. Remember, a micro service implementation can change with time.
They have to provide technical governance so that the teams in their technical development follow principles of Microservice.
At times they work as custodians of overall Microservices architecture.

487. What is the advantage of Microservices architecture over Service Oriented Architecture (SOA)?

Service Oriented Architecture (SOA) is an approach to develop software by creating multiple services. It creates small parts of services and promotes reusability of software. But SOA development can be slow due to use of things like communication protocols SOAP, middleware and lack of principles.
On the other hand, Microservices are agnostic to most of these things. You can use any technology stack, any hardware/middleware, any protocol etc. as long as you follow the principles of Microservices.
Microservices architecture also provides more flexibility, stability and speed of development over SOA architecture.

488. Is it a good idea to provide a Tailored Service Template for Microservices development in an organization?

If your organization is using similar set of technologies, then it is a good idea to provide a Service Template that can be tailored by development teams. It can make development faster. Also it can help in promoting adoption of various good practices that are already built into template.
But if your organization uses wide variety of technologies, then it may not be wise to produce and maintain a template for each service. Instead of that, it is better to introduce tools that help in maintaining same set of practices related to Microservices among all such technologies.
There are many organizations that provide tailored templates for Microservices. Eg. Dropwizard, Karyon etc. You can use these templates to make faster development of services in your organization.
Also remember that template code should not promote shared code.
This can lead to tight coupling between Microservices.

489. What are the disadvantages of using Shared libraries approach to decompose a monolith application?

You can create shared libraries to increase reuse and sharing of features among teams. But there are some downsides to it.
Since shared libraries are implemented in same language, it constrains you from using multiple types of technologies.
It does not help you with scaling the parts of system that need better performance.
Deployment of shared libraries is same as deployment of Monolith application, so it comes with same deployment issues.
Shared libraries introduce shared code that can increase coupling in software.

490. What are the characteristics of a Good Microservice?

Good Microservices have these characteristics:

1. Loose coupling: A Microservice knows little about any other service. It is as much independent as possible. The change made in one Microservice does not require changes in other Microservices.
2. Highly cohesive: Microservices are highly cohesive so that each one of them can provide one set of behavior independently.
3. Bounded Context: A Microservice serves a bounded context in a domain and communicates with rest of the domain by using an interface for that Bounded context.
4. Business Capability: Microservices individually add business capability that is part of big picture in organization.

491. What is Bounded Context?

A bounded context is like a specific responsibility that is developed within a boundary. In a domain there can be multiple bounded contexts that are internally implemented. Eg. A hospital system can have bounded contexts like- Emergency Ward handling, Regular vaccination, Out patient treatment etc. Within each bounded context, each sub-system can be independently designed and implemented.

492. What are the points to remember during integration of Microservices?

Some of the important points to remember during integration of Microservices are:
Technology Agnostic APIs: Developing Microservices in a technology agnostic way helps in integration of multiple Microservices. With time, the technology implementation can change but the interface between Microservices can remain same.

Breaking Changes: Every change in Microservice should not become a breaking change for client. It is better to minimize the impact of a change on an existing client. So that existing clients' do not have to keep changing their code to adapt to changes in a Microservice.

Implementation Hiding: Each Microservice should hide its internal implementation details from another one. This helps in minimizing the coupling between Microservices that are integrated for a common solution.

Simple to use: A Microservice should be simple to use for a consumer, so that the integration points are simpler. It should allow clients to choose their own technology stack.

493. Is it a good idea for Microservices to share a common database?

Sharing a common database between multiple Microservices increases coupling between them. One service can start accessing data tables of another service. This can defeat the purpose of bounded context. So it is not a good idea to share a common database between Microservices.

494. What is the preferred type of communication between Microservices? Synchronous or Asynchronous?

Synchronous communication is a blocking call in which client blocks itself from doing anything else, till the response comes back. In Asynchronous communication, client can move ahead with its work after making an asynchronous call. Therefore client is not blocked.
In synchronous communication, a Microservice can provide instant response about success or failure. In real-time systems, synchronous service is very useful. In Asynchronous communication, a service has to react based on the response received in future.
Synchronous systems are also known as request/response based.
Asynchronous systems are event-based.
Synchronous Microservices are not loosely coupled.

Depending on the need and critical nature of business domain, Microservices can choose synchronous or asynchronous form of communication.

495. What is the difference between Orchestration and Choreography in Microservices architecture?

In Orchestration, we rely on a central system to control and call various Microservices to complete a task. In Choreography, each Microservice works like a State Machine and reacts based on the input from other parts.
Orchestration is a tightly coupled approach for integrating Microservices. But Choreography introduces loose coupling. Also, Choreography based systems are more flexible and easy to change than Orchestration based systems.
Orchestration is often done by synchronous calls. But choreography is done by asynchronous calls. The synchronous calls are much simpler compared to asynchronous communication.

496. What are the issues in using REST over HTTP for Microservices?

In REST over HTTP, it is difficult to generate a client stub.
Some Web-Servers also do not support all the HTTP verbs like-GET, PUT, POST, DELETE etc.
Due to JSON or plain text in response, performance of REST over HTTP is better than SOAP. But it is not as good as plain binary communication.
There is an overhead of HTTP in each request for communication.
HTTP is not well suited for low-latency communications.
There is more work in consumption of payload. There may be overhead of serialization, deserialization in HTTP

497. Can we create Microservices as State Machines?

Yes, Microservices are independent entities that serve a specific context. For that context, the Microservice can work as a State Machine. In a State Machine, there are lifecycle events that cause change in the state of the system.
Eg. In a Library service, there is a book that changes state based on different events like- issue a book, return a book, lose a book, late return of a book, add a new book to catalog etc. These events and book can form a state machine for Library Microservice